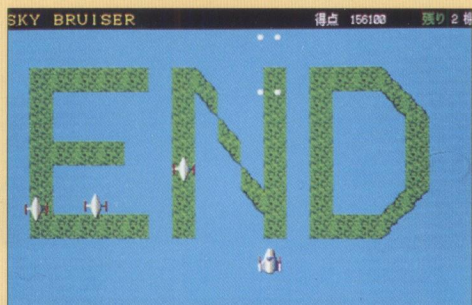
PC-9801シリーズ

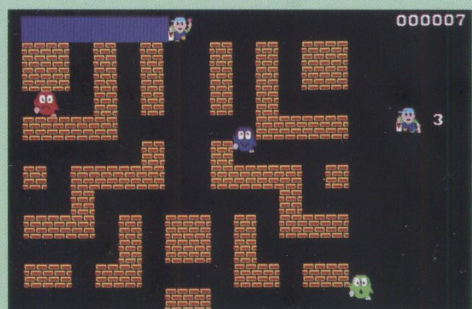
マシン語ゲームプログラミング

青山 学，日高 徹 共著

アスキー出版局

① プレイ・ザ・ゲーム（本書に掲載されているゲームの各場面）

●スカイ・ブルーザー（6章より）

●シューティング・ゲーム（2章，3章より）

●ペンキ・ボーイ（4章，5章より）

スカイ・ブルーザー（6章スクロールゲーム参照）

西歴ADC08年　思考のシステム化，感覚の絶対数値化理論により世界最大のネットワークマフィアとなったTH社．このTH社のおかげで，常にNo.2でしかないGI社．このGI社の常務であるあなたは，No.1になるためには世論を味方にするのが一番と判断した．そこで，思考システムや数値化は，人間の尊厳を傷つけるものであるという運動を推進した．これがもとで，ついに会社間戦争となる．

GI社常務のあなたは，旧式だがセミオート照準を持つクテノポマIIIを操り，TH社の奥深くにあるストラクチャー・ボールを破壊することにした．このストラクチャー・ボールこそTH社を運営しているエキスパート・システムなのだ．

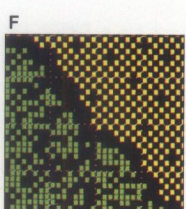
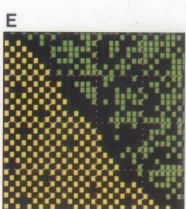
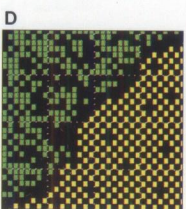
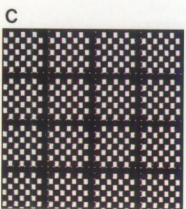
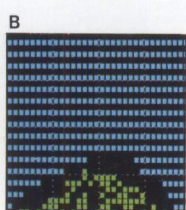
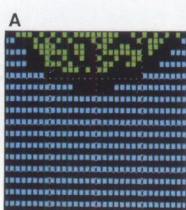
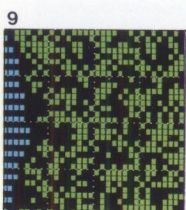
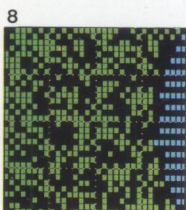
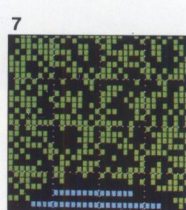
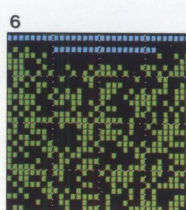
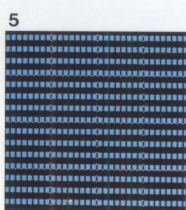
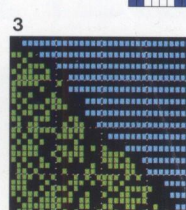
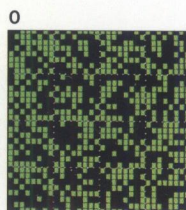
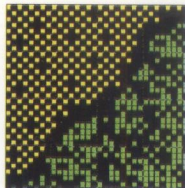
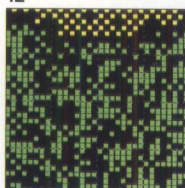
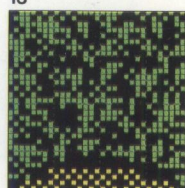
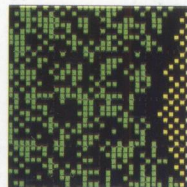
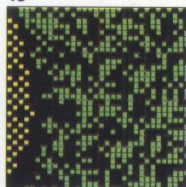
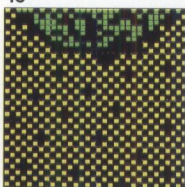
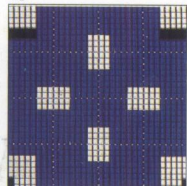
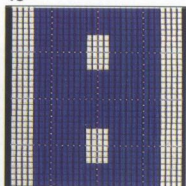
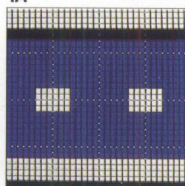
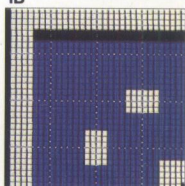
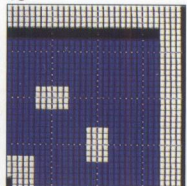
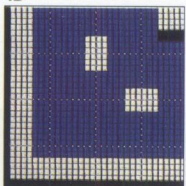
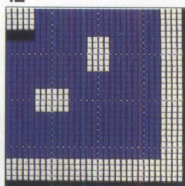
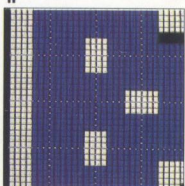
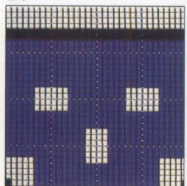
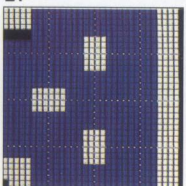
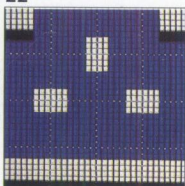
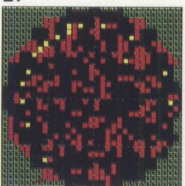
●スクロール・ゲーム用キャラクタ…地上パターン（6章参照）

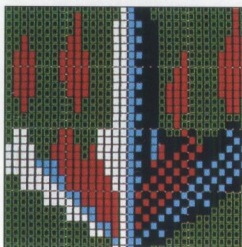
弾

0　1　2　3

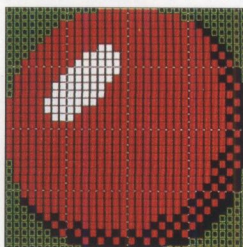
4　5　6　7

8　9　A　B

C　D　E　F

●爆発パターンは敵戦闘機のパターンに続けて登録(ただし10～1Fは予備として空けておく)

③ スカイ・ブルーザー用パターン

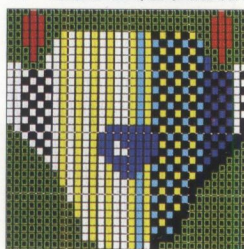
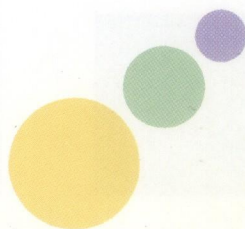
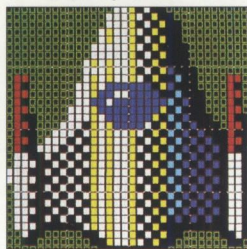
9：傘〔飛行帆船〕

●なぜ，こんなものが飛んでくるのか分からない．一説によると，ゴルフ好きのエンジニアが，仕事があまりに忙しいため好きなゴルフができないと怒り，新型の飛行帆船をこんな形にしてしまったとのことである．

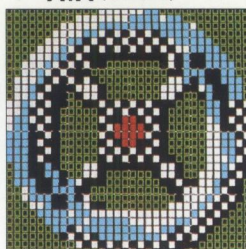
3：アンサーテ〔対空要塞

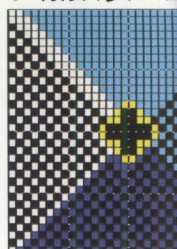
8：パープル・ベネディクション〔無人気球〕

●ベネディクション同様，放っておくと拡散するミサイルの雨を降らせる．うずまき軌道を描き始めたら，そろそろである．

A：カラシン〔迎撃戦闘機〕

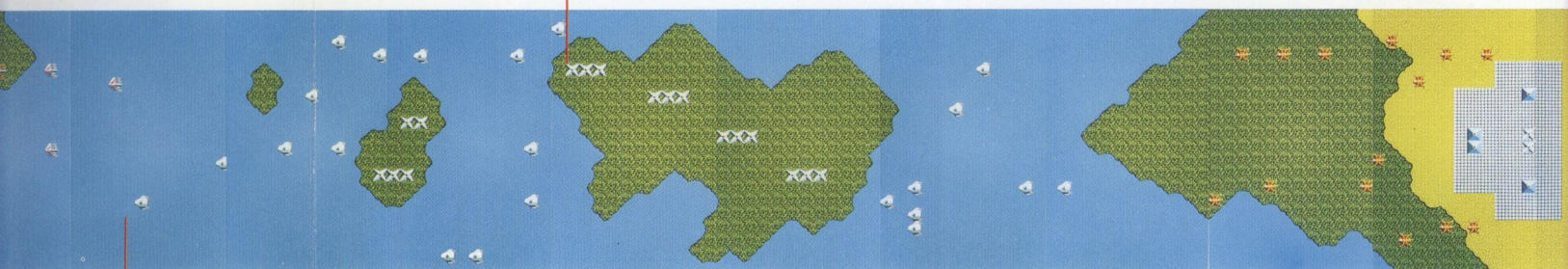
●追跡装置
るため，か
難のわざで

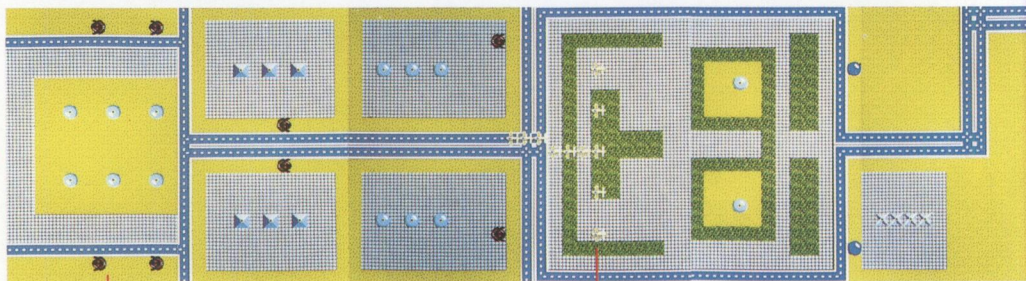
0：クテノポマIII〔メタフィジック・戦闘機〕

●旧式な戦闘機で，シールドが弱く一発のプロトン・ミサイルで破壊されてしまう．しかし，最高速度，ミサイルの搭載量では，最新型のものを上まわる．

4：XIX〔戦闘機〕

●別名を「運命の輪」もしくは，「The World」と呼ばれている軽戦闘機，編隊で飛行していることが多い．したがってパイロットはみな変態である．

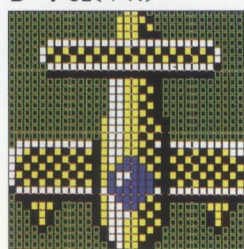
1：カルバリ〔TH社

3：アンサーテ〔対空要塞〕

●星型をした要塞で，内部にヒランヤ・コンピュータを備えており，同じ要塞でもセルティクより数段危険な存在である．

START

カラシン〔迎撃戦闘機〕

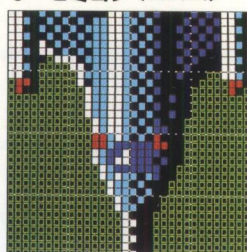
●追跡装置を持っているため，かわすのは至難のわざである．

C：イシズカン〔迎撃機〕

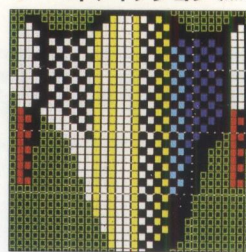
●ただひたすら近よってくる．この不気味な機体にみとれているあなた，そのままだと押し潰されてしまいますよ．

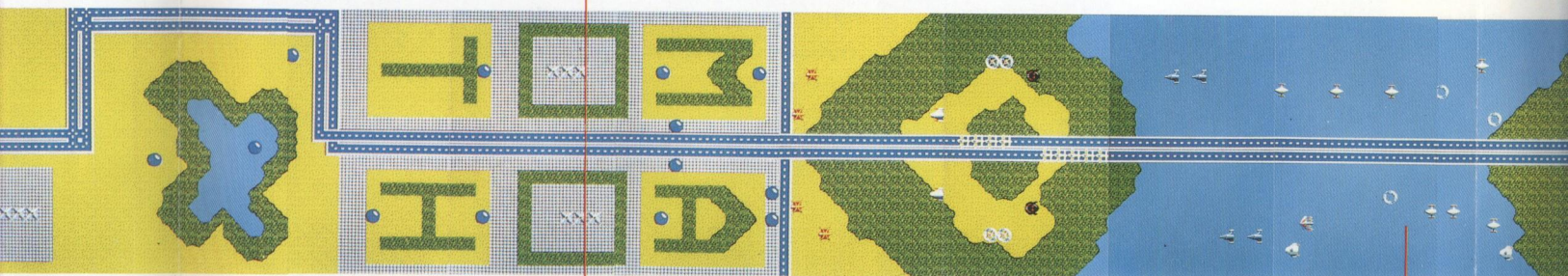
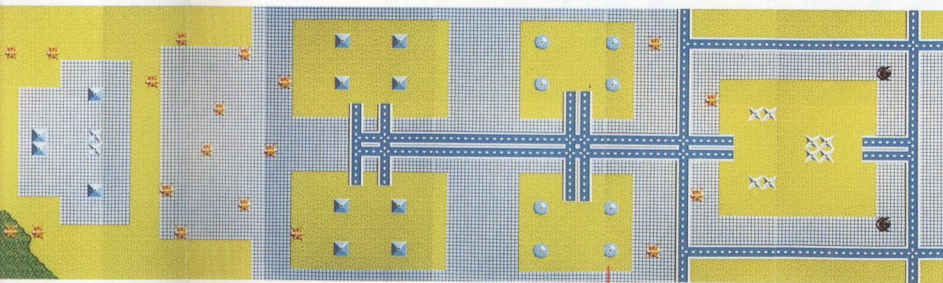
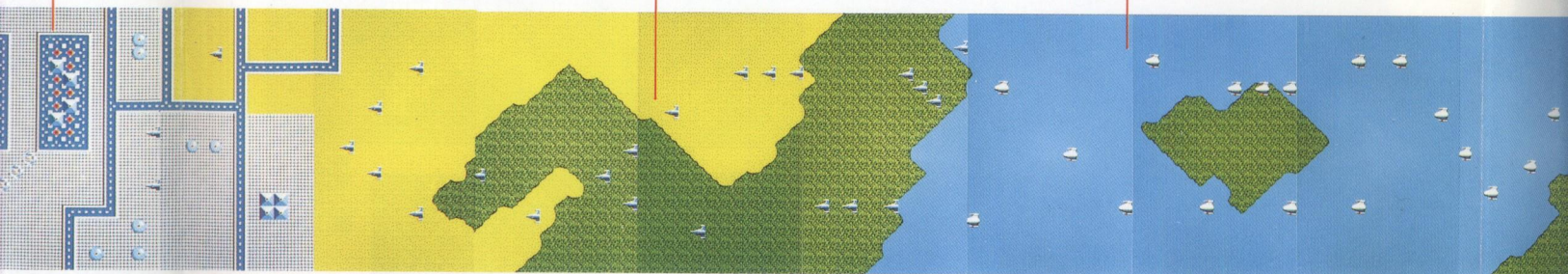
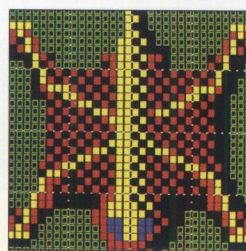
D：F52〔不明〕

●そうとう旧式の戦闘機で，情報や資料のたぐいはまるでない．

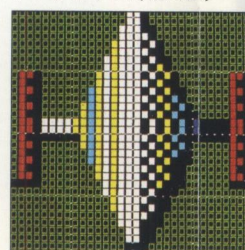
カルバリ〔TH社の倉庫〕

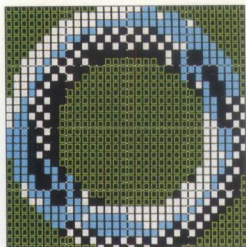
●TH社の製品，市場操作のために買占めた物資をしまっておくための倉庫，倉庫と言っても最低限の武装はしているので注意は必要．

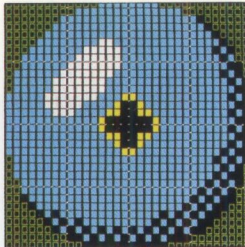
5：セミヨン〔戦闘機〕

●別名「キンメダイ」と呼ばれるあまりおめでたくない有人戦闘機．

6：ベネディクション〔無人飛行船〕

●祝福という名前をもつ飛行船．放っておくと拡散するミサイルの雨という祝福が与えられる．

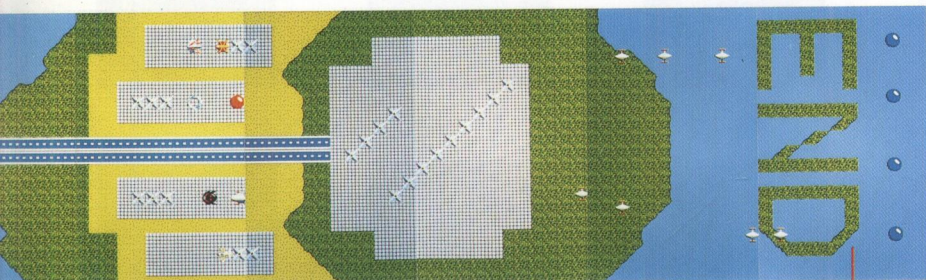
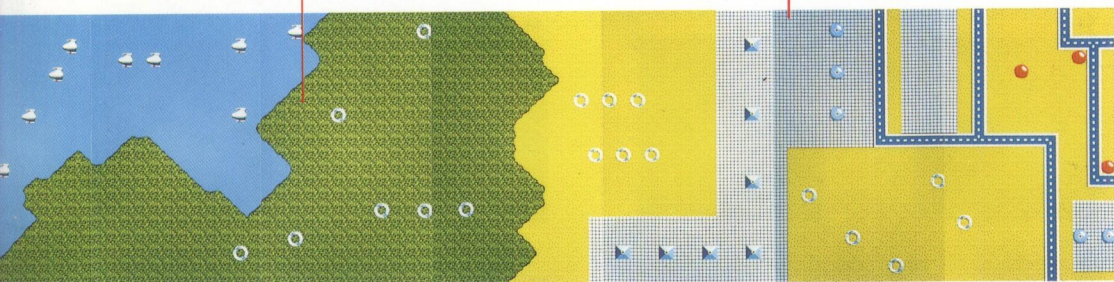
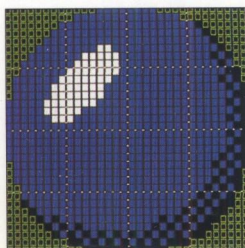
式の戦闘
資料のた
ない．

B：エクロール・バラン〔迎撃戦闘機〕

●無人の迎撃戦闘機で，別名「モモンガ」と呼ばれている．このたちの良くないモモンガに出会ったら，必ずミサイルで御あいさつをするように．

F：ケルテス〔戦闘機〕

●最新
ールド
手ごわ
発や2
は爆発
サイル
の何倍
放出し

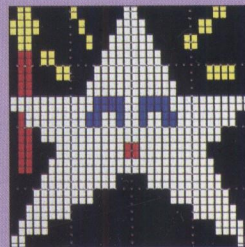
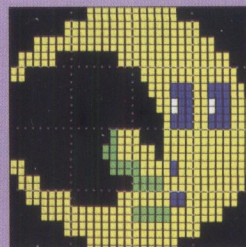
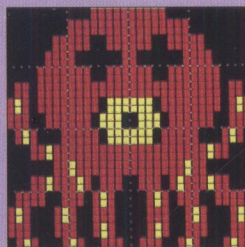
7：ガー〔戦闘機〕

●情報不足につき不明.

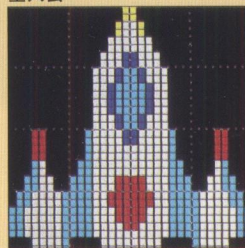
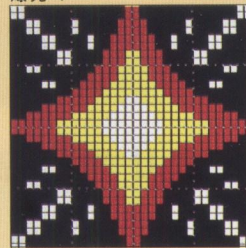
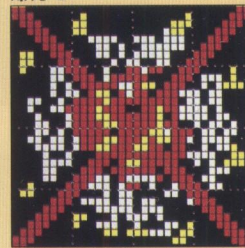
2：セルティック〔対空要塞〕

●無断で進入してくる敵を破壊するために作られた円形要塞，いまだに，ピラミッドパワーを利用しているらしい．そのためか，ミサイルを発射するタイミングがつかみにくい．

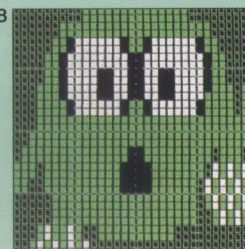
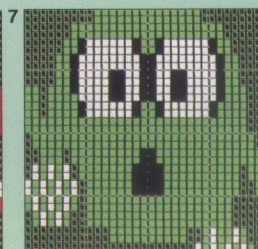
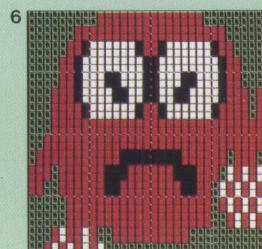
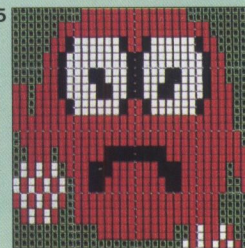
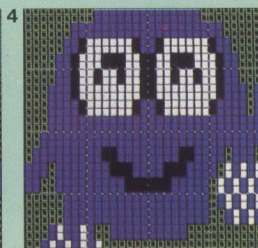
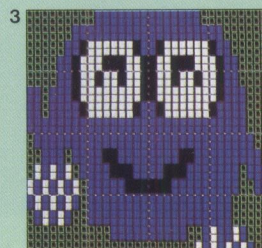
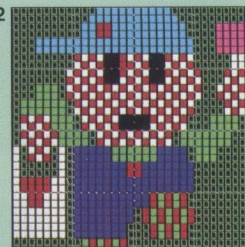
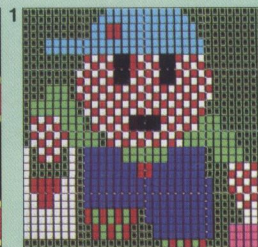
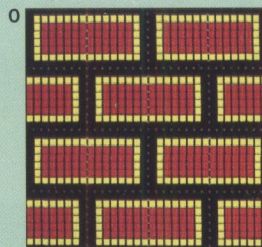
END

●最新のレーダーとシールドを有するかなり手ごわい戦闘機で，1発や2発のミサイルでは爆発しないうえ，ミサイルが当るたびにその何倍ものミサイルを放出してくる．

E：ストラクチャー・ボール〔浮遊物〕

●大気を構成している分子構造のゆらぎを利用して，空間に固定されているミサイル・ランチャー．

④ キャラクタ・パターン集

●シューティング・ゲームの敵キャラクタ
(2,3章参照)

●主人公と爆発,弾のキャラクタ
(2,3章参照)

主人公

爆発-1

爆発-2

弾

●迷路型ゲームのキャラクタ
(4,5章参照)

0

1

2

3

4

5

6

7

8

PC-9801シリーズ

マシン語ゲームプログラミング

青山 学, 日高 徹 共著

アスキー出版局

商　標

MS-DOS は米国 Microsoft 社の登録商標です．
その他本文中に登場する製品名は一般に各開発メーカーの商標です．なお，本文中では TM，Ⓡマークは明記していません．

本文イラスト　岩村　実樹

はじめに

本書は，ずばりマシン語でリアルタイムゲームを作りたいあなたに贈る，マシン語ゲーム制作『種あかしの秘本』です．この１冊の本の中に，マシン語ゲーム作りのノウハウがぎっしりと詰まっています……．

現在広く普及しているNECのPC-9801シリーズですが，日本製パソコンとしては初めて互換機が登場するなど，その地位はますます揺るぎないものとなっています．そして，この人気の一翼を担っているのは，紛れもなくマシン語ゲーム・ソフトです．しかし，本体付属のマニュアルは，BASICに関しては大変わかりやすく，またていねいに書かれているのですが，マシン語についてはどういうわけか，できることならやらないで欲しい，といわんばかりの内容でしか書かれていません．そのため，マシン語をマスターしたい人は，どうしても市販の書籍に頼らざるを得ないことになります．

マシン語に関する書籍は多数出版されており，また内容的にも立派なものばかりなのでケチなどつけようがありませんが，読んでみると読者の要求とはかなり違っているような気がします．つまり，マシン語は教えてくれても，ゲームのためのマシン語は教えてくれないのです．禅問答のような気がするかもしれませんが，入門者にとってこのギャップは実に大きな障害であり，ここで挫折していった人のことを思うと残念でなりません．

本書は，ゲームのためにマシン語をマスターしたいあなたが，回り道をすることなく目的を達成できるよう，マシン語の基礎からソフトハウス向けの高度なテクニックまで，そのすべてをわかりやすく解説したものです．その上，テーマはゲームとなっていますが，結果を画面で確かめながらマシン語をマスターできるため，マシン語の実践的な入門書としても十分役に立つ内容となっています．また，ゲーム制作の必需品であるパターンエディタも，本書用にオリジナルのものが提供されています．これらは，これまで閉ざされてきた暗闇のゲームマシン語の世界に，最初から灯りをつけて，そのすべてを公開しようという，アスキーならではの大胆かつ雄大な企画なのです．ぜひ，本書を極められて，アイディアあふれるおもしろいゲームを作りあげてください．

1991年１月　著者を代表して　日高　徹

本書の構成

本書は１章から６章までをステップアップ形式で読み進んでいくように作られています．各章に掲載したプログラムも，それ以前の章で作ったプログラムをもとに，さらに高度なものへと発展していきます．とくに2～4章のプログラムの多くは，5～6章で作られる迷路型ゲームとスクロール・ゲームで読み込まれるので，単体では動作しない部品という形をとっています．しかし，各章に用意したテスト・プログラムを使えば，部品プログラムの動作が確認できるようになっています．

各章の部品になるプログラムとテスト・プログラム対応関係は次のページの掲載プログラム一覧を参照してください．

本書は以下のシステムを対象としています．

対象機種　　PC-9801シリーズ（ただし初代PC-9801，UについてはVRAMの構成が異なるため変更が必要）
　　　　　　PC-98DO，DO＋（98モード）
　　　　　　PC-98XL，XL2，RL（ノーマルモード）
対象OS　　 MS-DOS Ver.2.11以降
アセンブラ　MASM Ver.4以降（Ver.6ではVer.5.1互換スイッチを使用すること）

＊本文中で言うVRAMとは，とくに断わりのない限り，グラフィックVRAMのことを指します．

掲載プログラム一覧

＊ファイル名の欄に“～.EXE”の表記のないものは部品プログラム

章	ページ	対応番号	ファイル内容	ファイル名	実行時に必要なファイル	アセンブル時に必要なファイル
1	36	リスト 1-1	線を引く	LIST1-1.ASM LIST1-1.EXE		
	45	リスト 2-1	豆腐の表示	LIST2-1.ASM LIST2-1.EXE		
	57	リスト 2-2	パターンの表示	LIST2-2.ASM LIST2-2.EXE	PTNDAT1.DAT	
	61	リスト 2-3	パターンの表示(高速版)	LIST2-3.ASM		
	65	テスト 2-3	テスト・プログラム	TEST2-3.ASM TEST2-3.EXE	PTNDAT1.DAT	LIST2-3.ASM
	69	リスト 2-4	パターンの部分消去	LIST2-4.ASM		LIST2-3.ASM
2	72	テスト 2-4	テスト・プログラム	TEST2-4.ASM TEST2-4.EXE	PTNDAT1.DAT	LIST2-4.ASM
	75	リスト 2-5	データによるパターンの移動	TEST2-5.ASM TEST2-5.EXE	PTNDAT1.DAT	LIST2-4.ASM
	80	リスト 2-6	パターンの大量出現	LIST2-6.ASM		LIST2-4.ASM
	82	テスト 2-6	テスト・プログラム	TEST2-6.ASM TEST2-6.EXE	PTNDAT1.DAT	LIST2-6.ASM
	89	リスト 2-7	キー入力による移動と玉の発射	LIST2-7.ASM		LIST2-6.ASM
	93	テスト 2-7	テスト・プログラム	TEST2-7.ASM TEST2-7.EXE	PTNDAT1.DAT	LIST2-7.ASM
	98	リスト 3-1	衝突の判定	LIST3-1.ASM		LIST2-7.ASM
	102	リスト 3-2	文字の表示	LIST3-2.ASM		LIST3-1.ASM
	104	テスト 3-2	テスト・プログラム	TEST3-2.ASM TEST3-2.EXE	PTNDAT1.DAT MOJI.DAT	LIST3-2.ASM
	107	リスト 3-3	得点の計算と表示	LIST3-3.ASM		LIST3-2.ASM
	107	テスト 3-3	テスト・プログラム	TEST3-3.ASM TEST3-3.EXE	PTNDAT1.DAT MOJI.DAT	LIST3-3.ASM
3	111	リスト 3-4	得点の計算と表示	LIST3-4.ASM		LIST3-3.ASM
	112	テスト 3-4	テスト・プログラム	TEST3-4.ASM TEST3-4.EXE	PTNDAT1.DAT MOJI.DAT	LIST3-4.ASM
	118	リスト 3-5	シューティング・ゲームの仕上げ	LIST3-5.ASM		LIST3-4.ASM
	121	テスト 3-5	テスト・プログラム	TEST3-5.ASM TEST3-5.EXE	PTNDAT1.DAT MOJI.DAT	LIST3-5.ASM
			テスト用パターンデータ	PTNDAT1.DAT		
			文字パターンデータ	MOJI.DAT		

章	ページ	対応番号	ファイル内容	ファイル名	実行時に 必要なファイル	アセンブル時に 必要なファイル
4	130	リスト 4-1	BEEP 音楽の演奏	LIST4-1.ASM		
	131	テスト 4-1	テスト・プログラム	TEST4-1.ASM TEST4-1.EXE		LIST4-1.ASM
	135	リスト 4-2	BEEPによる効果音	LIST4-2.ASM		LIST4-1.ASM
	136	テスト 4-2	テスト・プログラム	TEST4-2.ASM TEST4-2.EXE		LIST4-2.ASM
	146	リスト 4-3	FM音源によるハープシコード演奏	LIST4-3.ASM LIST4-3.EXE		
5	156	リスト 5-1	迷路の表示	LIST5-1.ASM		
	165	テスト 5-1	テスト・プログラム	TEST5-1.ASM TEST5-1.EXE	MEIRO.DAT	LIST5-1.ASM
	173	リスト 5-2	テンキーによる移動	LIST5-2.ASM		LIST5-1.ASM
	183	テスト 5-2	テスト・プログラム	TEST5-2.ASM TEST5-2.EXE	MEIRO.DAT MOJI.DAT	LIST5-2.ASM
	188	リスト 5-3	敵の移動と追跡	LIST5-3.ASM		LIST5-2.ASM
	194	テスト 5-3	テスト・プログラム	TEST5-3.ASM TEST5-3.EXE	MEIRO.DAT MOJI.DAT	LIST5-3.ASM
	197	リスト 5-4	ペンキ・ボーイの仕上げ	LIST5-4.ASM		LIST5-3.ASM
	199	テスト 5-4	テスト・プログラム	TEST5-4.ASM TEST5-4.EXE	MEIRO.DAT MOJI.DAT	LIST4-2.ASM LIST5-4.ASM
			迷路用パターンデータ	MEIRO.DAT		
6	206	リスト 6-1	重ね合わせ処理	LIST6-1.ASM		
	213	テスト 6-1	テスト・プログラム	TEST6-1.ASM TEST6-1.EXE	SKYBRU.DAT	LIST6-1.ASM
	221	リスト6-2a	スクロール処理	LIST6-2.ASM		LIST6-2B.ASM
	233	リスト6-2b	スクロール処理	LIST6-2B.ASM		
	247	テスト 6-2	テスト・プログラム	TEST6-2.ASM TEST6-2.EXE	SKYBRU.DAT MAPPAT.DAT MAPDAT2.DAT	LIST6-2.ASM
	254	リスト 6-3	疑似ロボット言語	LIST6-3.ASM		LIST6-2.ASM
	273	テスト 6-3	テスト・プログラム	TEST6-3.ASM TEST6-3.EXE	SKYBRU.DAT MAPPAT.DAT MAPDAT2.DAT	LIST6-3.ASME
	280	リスト 6-4	スカイ・ブルーザーの仕上げ	TEST6-4.ASM TEST6-4.EXE	MOJI.DAT SKYBRU.DAT MAPPAT.DAT MAPDAT2.DAT	LIST6-3.ASM
			スカイ・ブルーザー用パターン	SKYBRU.DAT		
			スカイ・ブルーザー用マップパターン及びマップデータ	MAPPAT.DAT MAPDAT2.DAT		
A	304		パターン・エディタ	MSPTER.EXE		
	311		マップ・エディタ	MAPEDIT.EXE		

Contents

6章 ●スクロール・ゲーム

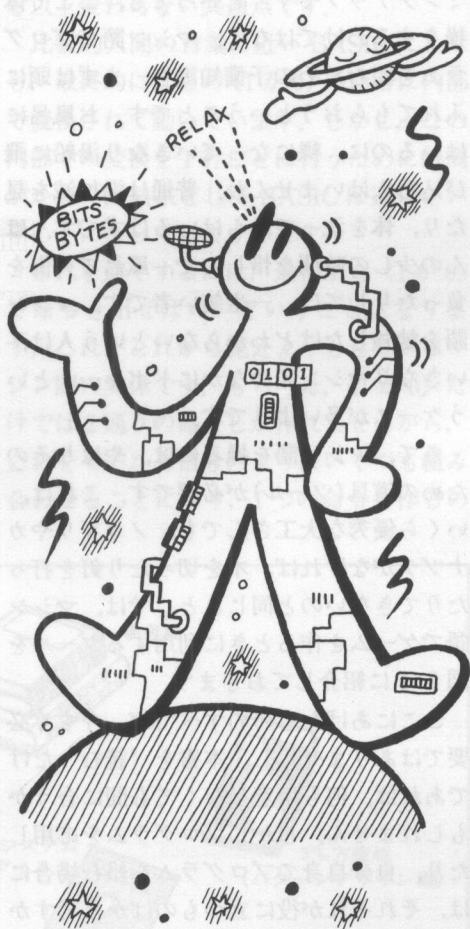
Appendix

数字・文字パターン

(3章参照)

1章 ●ウォーミング・アップ

●BASICに限界を感じ，マシン語を覚えようとしている今のその気持ち，最後まで大切にしてくださいネ．その気持ちさえ忘れなければ，もうマシン語なんてモノにしたも同然ですから，あせらずに気楽に進みましょう．何事もゆとりが肝心です．やさしいことも，あわてると難しく見えるものです．マシン語も同じです．あまり，難しく考えると途中で挫折してしまいます．でも，もし運悪く挫折してしまったら，そのときはお手紙ください．復活の魔法をかけてあげましょう．

●P.S.マシン語なんてやさしい!!　そう思ってけっこうです．ただし，すべてのマシン語プログラムがやさしいとは言いません．それは，BASICでも同じことでしょう．そこで，BASICを覚えたときのように，簡単なことでも1つひとつ確認をしながら，その内容を理解していけばいいのです．どうか，1週間や2週間で本書の内容を読破しようなどというハリキリ精神は捨ててください．…挫折のもとです．あわてなくても，ゴールは1ページずつこちらに近づいてきます．

1. 小道具…これだけはそろえておこう

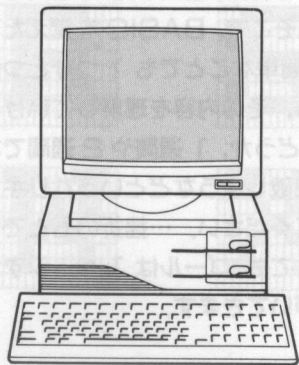
「さあ，マシン語をマスターするぞ!!」と，期待して本書を開いた方はガッカリするかもしれませんが，まずは肩ならし，ウォーミングアップです．といっても，ここで体操をするわけではなく，マシン語でプログラムを組むための予備知識を，まずは頭に入れてもらおうということです．お風呂にはいるのに，裸になっていきなり湯船に飛び込む人はいませんね．普通は湯加減を見たり，体を洗ってからはいるはずです．ほんの少しの時間を惜しんで，風呂で火傷を負ったりしては，一生笑い者です．マシン語を勉強したけどわからないという人は，いきなりマシン語のなかにドボン……というケースが多いようです．

さて，マシン語を操るには，やはりそのための道具(ツール)が必要です．これは，いくら優秀な大工さんでも，ノコギリやカナヅチがなければ，木を切ったり釘を打ったりできないのと同じこと．では，マシン語でゲームを作るときに利用するツールを図1-1に紹介しておきます．

ここにあげたもののすべてが，今すぐ必要ではありませんし，本書を〝読む〟だけであれば，何も用意しなくても間に合うかもしれません．しかしプログラムを応用したり，自分自身でプログラムを組む場合には，それぞれが役に立つものばかりですから，財布と相談しながら手に入れるようにしてください．

アセンブラに関しては，本書ではMS-DOS上のマクロアセンブラ『MASM』を利用することを前提に話を進めていきますが，他のものでも問題はありません．ただし，本書のサンプルプログラムでは，MASM独自のマクロ命令などを使っているため注意が必要です．MASMはVer.4以降のものを対象としています．

また，MASMを使うということで，当然MS-DOSのシステムディスクが必要になります．これはVer.2.11以降のものを用意してください．

参考書としては，『新版 PC-9800シリーズ テクニテクニカルデータブック』，『はじめて読むMASM』(共にアスキー出版局発行)，『8086マシン語秘伝の書』(啓学出版発行)などがお奨め品です．また，ここにあげた本以外にも，自分にあったものを本屋さんで探してみてください．

PC-9801シリーズのコンピュータ

2. 数…2進数と16進数

コンピュータは人間が必要に迫られて作り出したものですから，そこには当然のことながら，人間にはできない能力が秘められています．それが計算の速度であり，正確さです．お隣の国，中国ではコンピュータのことを「電脳」というそうですが，これには何となく人間臭さを感じて親しみが湧いてきます．まるで人間の頭に電気を通したみたいですが，人間の頭脳とコンピュータの頭脳とのいちばん大きな違いは何かといえば，コンピュータには大体とか，適当にという感覚がないことです．中庸などはないのです．つまり何でも白か黒かはっきりさせるということです．

この《アルかナイか》を数字で表現すると《1か0》になります．これが2進数の基本です．そしてコンピュータは，この《1か0》を電気が通っているいないかで処理するのです．これは，現在のところどんなメーカーのどんな機種でも，コンピュータである以上変わらない共通点です．

比較的人間の言葉に近いBASICにしても，最終的にはこの《1か0》の命令に内部で変換されて動いています．しかし，この内部での変換を1行ごとに行うために時間がかかり，結果としてBASICは遅いということになってしまうのです．

これを速くするには，最初から《1か0》で命令を出せばいいということになります．これがこれから覚えようとしているマシン語の実体です．もちろん，《1か0》だけでは2通りの命令しか作れませんから，これをモールス信号のようにいくつも組み合わせることにより，1つの命令を作るの

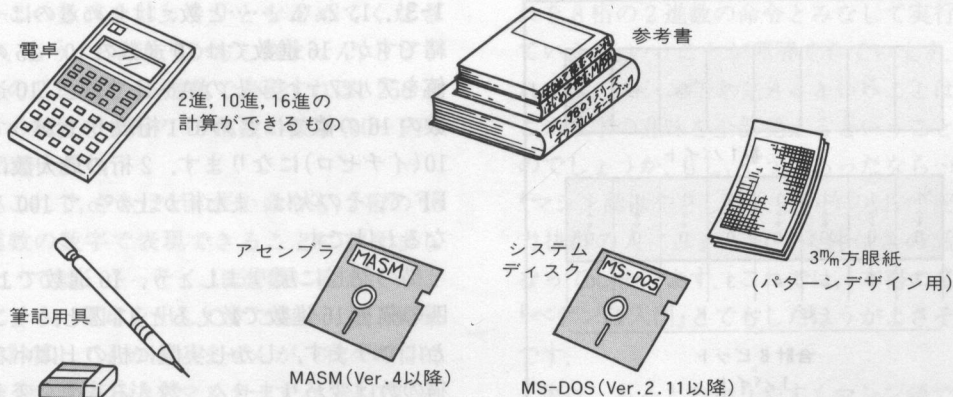
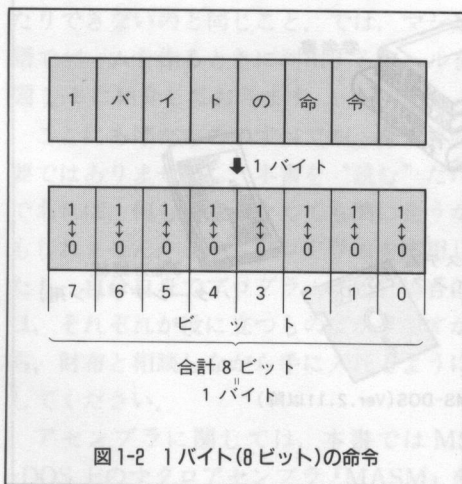



図1-1　マシン語でゲームを作るためのツール

です．そうすれば，《1か0》だけでもたくさんの命令を作れることになります．たいへんなことのようですが，むずかしく考える必要はありません．この《1か0》で作った命令を暗記しようというわけではないのですから……．

ここでは，命令の基本となる《1か0》を数えるのに，**ビット**と呼ばれる単位を用います．ですから《1か0》が2つならば2ビット，5つならば5ビットということです．しかし命令によって2ビットを使ったり5ビットを使ったりするのでは，いくらコンピュータでも処理しにくいですね．だいいち，どこまでが1つの命令なのかわかりません．そこで8ビットを1セットにして命令を表すことにしたのです．そして，この1セットつまり8ビットのことを**1バイト**と呼びます．もちろん1つの命令は1バイトとは限らず，複数のバイトで構成されるものもあります．

ここで，図1-2を見てください．

図1-2 1バイト(8ビット)の命令

マシン語の命令が8ビットを単位として組立られていることは先に述べましたが，例として，《1か0》の8つの組み合せでできた1つの命令を書いてみます．

11000011

どうですか．これはもう立派なマシン語の命令です．しかし，いくら8ビットを単位にコンピュータが処理してくれるといっても，これではあまりに長過ぎます．それにこんな命令では命令をする我々のほうがたまりません．そこで，まず短く表すことから考えてみましょう．長くなった原因は，命令を2進数，つまり1と0だけで書いているからです．数える基準をちょっと変えれば，もっと短くてすむはずです．

たとえば，机の上に猫が17匹こっちを向きゴロニャンしているとします．この猫を，もし16進数で数えたらどうでしょうか．

なぜ，16進数かという説明は後にして，まず16進数の数え方を覚えてください(図1-3)．1，2，3，……と数えはじめるのは一緒ですが，16進数では10進数の10〜15の値をアルファベットのA〜Fで表し，10進数の16の値ではじめて1桁繰り上がって10(イチゼロ)になります．2桁の最大数はFFで，その次は，また桁が上がって100となるわけです．

猫の話しに戻りましょう．10進数で17匹の猫を16進数で数えると11匹ということになります．しかし実際に机の上にいる猫の数は変わりません．数える基準を変えただけで猫そのものには何もしていないのですから，これは当り前のことです．命令

図 1-3　16 進数の数え方

を短くし，かつコンピュータの処理のしやすさも考えると，数値表現にはこの16進数が一番都合がいいのです．

ここで8ビット（8桁の2進数）で表現できる命令の数を調べてみましょう．1ビットにつき《1か0》の2通りの表現ができますから，次のように計算できます．

8ビットで表現できる命令
＝2×2×2×2×2×2×2×2通り
＝　16　×　16　通り
＝256通り

この数式をじっくりとながめてください．もう，おわかりかもしれません．8ビットの2進数を16進数で表現すれば，256通りの数値を2桁の16進数の数字（00〜FF）で表せます．

これで，さきほどの長い命令も，2桁の16進数の数字で表現できることになりました．では，さっそく16進数に変えてみましょう．電卓を出してください．紙と鉛筆で計算してもかまいませんが，我々の目的はその計算方法をマスターすることではありません．ここは，結果だけを求めて，さらりと通り過ぎてしまいましょう．

まず，2進数のモードにして11000011と入力します．そして，16進数のモードに変換します．C3と表示されています．

11000011 ＝ C3

どうです．だいぶスッキリしましたね．コンピュータには，この2桁の16進数で命令すればいいのです．そして，これが我々が作ろうとしているマシン語の命令なのです．

もう，あなたは「マシン語とは2桁の16進数（00〜FF）のことで，コンピュータはそれを8桁の2進数の命令とみなして実行している」ということが理解できています．それでは，マシン語を覚えるということは，この数字の意味を全部覚えるということなのでしょうか．もし，そうであったなら……『マシン語はやさしい』というのは，『記憶力抜群の人には』という条件付きの話になってしまいます．これでは，本書の名も『ペテン語入門』とでもしたほうがよさそうです．

実は，もっとわかりやすくマシン語でプログラムが作れます．そのためにアセンブラが必要なのです．

3. アセンブラ…マシン語開発ツール

マシン語の命令とは，いったいどんな内容だと思いますか．かなりいろいろな意味を持った命令がありそうですね．ところが，実際は非常につまらないことしか命令できないのです．簡単にいえば，数字をもてあそぶだけなのです．それも2つの数を足したり引いたり，どっちが大きいか比べたり，メモリのどこかに数字を置いてみたり……もちろんプログラムですから，比較した結果でBASICのGOTO文のようにどこかへジャンプすることもあります．それでも，いった先でまた同じように数字をいじっているだけなのです．

この程度なら，簡単に覚えられそうな気がしませんか．しかし，次の命令を見てください．左側の2桁の16進数がマシン語です，右側がその意味です．AXとかBXとかいうのは変数と思ってください．

48 …… AXの内容から1を引く
(AX＝AX－1)
4B …… BXの内容から1を引く
(BX＝BX－1)
49 …… CXの内容から1を引く
(CX＝CX－1)
4A …… DXの内容から1を引く
(DX＝DX－1)

4つとも似たような内容なのに，マシン語の数字はバラバラです．その上，この数字を見ただけでは，引くとかAXとかBXとかを連想することはまったく不可能です．となると，ただ丸暗記するしか覚える方法はなさそうです．まあ，世の中には平気でこの数字でプログラムを組む人もいるらしいのですが，今はコンピュータの時代です．そんなめんどうなことは，コンピュータにまかせましょう．

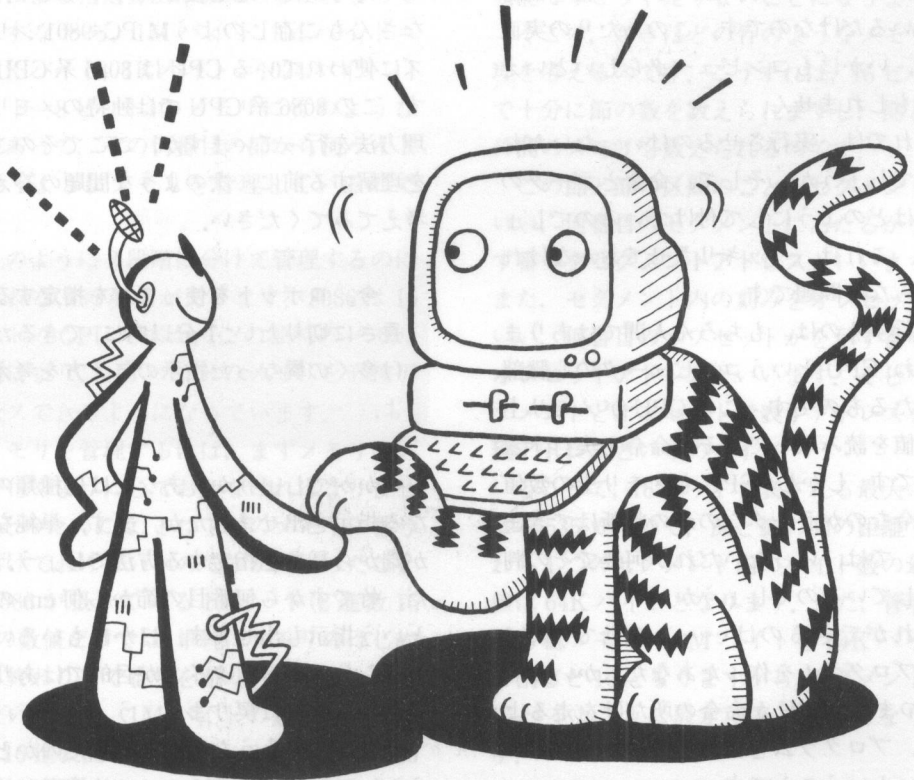
我々は，もう少しわかりやすい記号でこれらの命令を書いて，それをコンピュータで数字に変換してもらえばいいのです．この記号を数字に変換してくれるプログラムのことを**アセンブラ**といいます．そして，数字の代わりに我々が使う記号のことを**ニーモニック**というのです．ニーモニックはマシン語の数字を人間にわかりやすく記号化しただけで，その意味や内容はまったくマシン語と同じですから，これもマシン語と呼ばれます．あなたは，これから，このニーモニックのマシン語をマスターし，プログラムを組んでいくのです．

これで，なぜ貴重なお金を出してまでアセンブラが必要か，何となくわかったのではないかと思います．といいつつ，実はPC-9801には最初からアセンブラ機能が標準装備されているのです，……といったら怒るでしょうね．ただ，このオマケのアセンブラでは，長いマシン語プログラムを作るのが難しいため，だれもすすんで使ったりしません．

このアセンブラは，ワン・ライン・アセンブラなのでスクリーンエディットができません．つまり，BASICのように手軽にプログラムの修正や追加ができないのです．し

かもラベルが使えないとか，使用できない命令があるとか，およそ開発用のアセンブラとしては不適当といえます．

しかし，短いテスト・プログラムの作成や簡単な変更，あるいはプログラムの見直しなどには大変便利なものですので，その目的で利用すればそれなりに価値のあるものです．使用方法については，PC-9801本体付属のマニュアル(モニタの項)に詳しく書いてありますので，そちらのほうをお読みください．

本書ではこのモニタのアセンブラではなく，現在，広く使われているMS-DOS上のアセンブラであるMASMを使ってプログラムを組んでいきます．

4. メモリ管理…セグメントについて

マシン語の命令をコンピュータに実行させるということは，メモリ上に16進数の命令(00～FF)を置いて，それを実行させることだというのはすでに書きましたが，メモリとは文字どおりその命令などを記憶する場所のことです．

記憶するだけですから，00～FFの数値であれば，別に命令でなく何らかのデータでもかまわないのです．だいいち，メモリ自身は命令なのかデータなのか判断できません．ただ，1バイト(00～FF)の数を記憶しているだけなのです．このあたりの実直さは，いかにもコンピュータらしいといえるかもしれません．

それでは，実行させるのはいったいだれなのでしょうか．そして，命令とデータの区別はどのようにして付けているのでしょうか．これは，ハッキリさせておかなければならない問題です．

実行するのは，もちろん人間ではありませんね．CPUというコンピュータの心臓部にあたるものです．このCPUがメモリ上の数値を読み取って，その命令を実行するわけです．しかし，CPUもメモリ上の数値が命令なのかデータなのかの判断はできません．では，いったいだれが何処でその判断をしているのでしょうか．

それができるのは……この世でただ1人，プログラムを作ったあなたしかいません．つまり，CPUが命令の所だけを走るように，プログラムを組んでやらなければいけないということです．

もし，関係ないデータを命令として実行させたら……そのときは，まず，間違いなく暴走します．たいていは画面がメチャクチャになって，2度とキー入力ができなくなります．こうなったら素直にリセットする以外に道はありません．

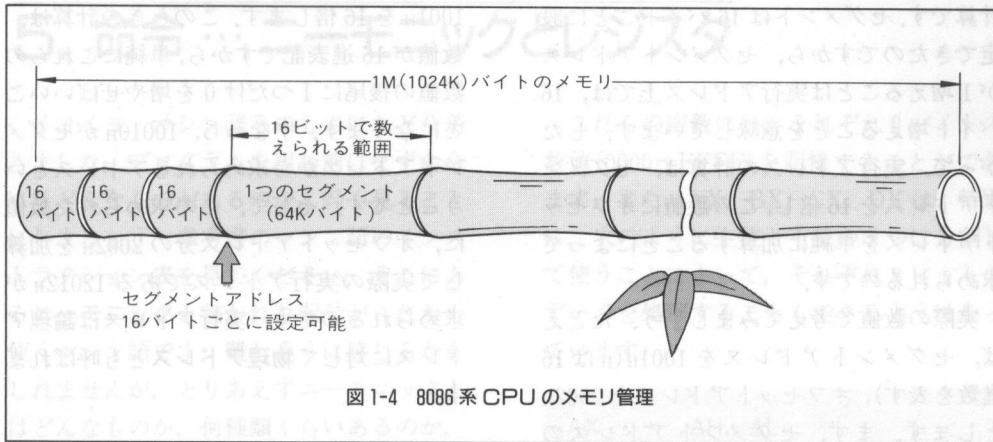
そこでCPUが暴走しないようなプログラムを組むには，メモリを我々がきちんと管理する必要が出てきます．

そのためには，PC-9801のメモリ管理について考えてみる必要がありそうです．みなさんもご存じのようにPC-9801シリーズに使われているCPUは8086系CPUです．この8086系CPUでは独特のメモリ管理方法を行っていますが，ここでそのことを理解する前に，次のような問題の答えを考えてみてください．

今，ロボットを使って竹を指定する長さに切りたい．1分以内に，できるだけ多くの異なった指示の与え方を考えよ．

いかがでしょうか，あなたは何種類の異なる指示を出せましたか．まず，単純なのが端から長さを指定する方法でしょう．また，竹ですから何番目の節から何cmの所という指示も出せます．ほかにもいろいろ考えられますが，クイズが目的ではありませんから本筋に戻りましょう．

なぜ，このような問題を出したのかというと，8086系のCPUのメモリ管理が後者

図1-4　8086系CPUのメモリ管理

の節による指定方法に似ているからなのです．すなわち，1Mバイト(1Mバイト＝1024Kバイト，1Kバイト＝1024バイト，1バイト＝8ビット)のメモリに竹のような節があって，その何番目の節から何バイトというようにメモリを管理しているのです．

このように2段階に分けて管理するのにはそれなりの理由があります．8086は16ビットのCPUですが，この独特のメモリ管理により，1Mバイトのメモリを直接アクセスできるようになっています．

メモリを管理するには，まずメモリの区別が付くようにしておかなければなりません．そのためには，1つ1つのメモリに番号を付けておけばいいのですが，困ったことに8086が扱っている16ビット(2進数16桁)の数値だけでは，単純にメモリのはじめから終わりまで番号を割り付けることができないのです．1Mバイトの数値を表すには，16進表記では5桁分，2進表記では20桁すなわち20ビット分必要ですから，2進表記で4ビット足りないことになります．

そこで，さきほどの竹のようなメモリ管理を考えるのです．こうすれば，16ビットで十分に節の数を数えられますし，節と節の間のメモリも数えられるのです．

この節と節の区間のことを**セグメント**といい，何番目のセグメントにあたるかを示す番号を**セグメントアドレス**といいます．また，セグメント内の刻みを**オフセット**といい，何番目のオフセットかを示す番号を**オフセットアドレス**といいます．そしてセグメントとオフセットで表すアドレスを**論理アドレス**というのです．

ところで，16ビットで表される最大の数は65535ですので，節と節の間の距離すなわち1つのセグメント内のバイト数の最大値は64Kバイトとなります．また，管理できる総メモリの1Mバイトを64Kバイトで割ると16となりますから，セグメントは16バイトごとに設定できることになります．

ここでわかりにくいのは実行アドレスの

計算です．セグメントは16バイトごとに設定できたのですから，セグメントアドレスが1増えることは実行アドレス上では，16バイト増えることを意味しています．したがって，実行アドレスの計算は，セグメントアドレスを16倍し，この数値にオフセットアドレスを単純に加算することによって求められるのです．

実際の数値で考えてみましょう．たとえば，セグメントアドレスを1001H（Hは16進数を表す），オフセットアドレスを2002Hとします．まず，セグメントアドレスの1001Hを16倍します．このときの計算は，数値が16進表記ですから，単純にこれらの数値の後尾に1つだけ0を増やせばいいことになります．すなわち，10010Hがセグメントアドレスから求められるアドレスということです．そして，この求められた数値に，オフセットアドレス分の2002Hを加算して実際の実行アドレスである12012Hが求められるのです．実行アドレスは論理アドレスに対して**物理アドレス**とも呼ばれます．

セグメント1001H，オフセット2002Hの場合の実行アドレスの計算

	値	説明
	10010H	…… セグメントアドレスを16倍した数
＋	2002H	…… オフセットアドレスを加算する
	12012H	…… 最終的な実行アドレスが求められる

一般的にセグメントアドレスとオフセットアドレスはコロン（：）で区切って，〝1001：2002H〟のように表記します．

全体のメモリ管理や，メモリマップの詳細などについては，最初に紹介した『新版PC-9800シリーズ　テクニカルデータブック』などに詳しく説明されていますので，ここでは基本的なメモリ管理の方法を述べるだけにします．マシン語の準備ばかりで，なかなか本筋にはいらないと，せっかくのあなたのヤル気がなくなってしまうかもしれませんからね…….

5. 命令…ニーモニックとレジスタ

イヨイヨ，マシン語そのものにたどり着きました．プログラムを組む前にまず命令にはどんなものがあるのか，軽く見ることにしましょう．参考書のマシン語のインストラクション表を見てください．そこにある**ニーモニック**と書かれた記号がこれから使うマシン語です．難しそうに感じるかもしれませんが，とりあえずニーモニックとはどんなものか，何種類くらいあるのか，それだけでも確認してください．今までニーモニックとは，「我々にわかりやすい記号」とだけしか書きませんでしたが，この表からはその記号が○とか△ではなくアルファベットであるということがわかります．そして，実はこのアルファベットは英語の単語を省略したものなのです．このことがニーモニックが人間にわかりやすいという理由なのです．ここで，次の文字を覚えてください．

```
AX  BX  CX  DX
DI  SI  CS  DS  ES
```

これはマシン語で使える変数です．マシン語の場合，BASICのように手軽に変数を作ることはできません．しかし，この9つでもうまくヤリクリすれば何とかなるものなのです．といっても，CSには，プログラムが置かれているセグメント値があらかじめ格納されており，書き換えることはまずありませんので，実質的には8つということになります．

これらの変数には，それぞれ2バイトの数値(0000～FFFF)を記憶することができます．また，AX，BX，CX，DXは，特別に，次のように上位と下位の2つに分割して使うことによって，それぞれ8ビットのデータを処理することもできるようになっています．

```
AX  →  AH, AL
BX  →  BH, BL
CX  →  CH, CL
DX  →  DH, DL
```

CPU内部には**レジスタ**と呼ばれるRAM(Random Access Memory)のようなものがあるのですが，実はこれらの変数の正体はそのレジスタなのです．

さて，この9つのレジスタは，一見，同格のように見えますが，大別すると**汎用レジスタ**，**インデックスレジスタ**，**セグメントレジスタ**の3つのグループに分かれます．AX，BX，CX，DXが演算や，データ処理に使われる汎用レジスタと呼ばれるものであり，SI，DIがメモリのオフセットの基準となるアドレス用で，インデックス・レジスタといいます．そして，CS，DS，ESがセグメントアドレス用のセグメントレジスタというわけです．本当はこのほかにもレジスタと呼ばれるものはあるのですが，今はこれだけ覚えてください．

マシン語の命令は，そのほとんどがレジスタに関係があります．ということは，ま

ずレジスタに数値を代入する命令を知らなければなりません．

```
MOV  AX, 0A1B2H
```

これは，AXに16進数の〝A1B2〟を代入するという意味です．〝MOV〟はMOVEの略です．

〝A1B2〟の前後に〝0〟と〝H〟がついていますが，これはMASMにおいて16進数を表記するときの決まりで，数字の最後にはHを，また数値がA～Fで始まる場合には頭に0を付けなければなりません．本書でも，MASMの文法にそって，ここから先は16進数の最後に$_{\mathrm{H}}$(16進数：Hexadecimal)を付けることにしますが，頭の0は本文中では邪魔なのでプログラムにだけ付けるようにしました．また，セグメントレジスタ以外は同じ書き方で数値を直接代入できます．

```
例
MOV  BX, 1234H    ; BXに1234Hを代入する
MOV  DX, 0F300H   ; DXにF300Hを代入する
MOV  AL, 0A1H     ; ALにA1Hを代入する
MOV  CH, 12H      ; CHに12Hを代入する
```

ニーモニック中のスペースは1スペースあればいいのですが，そろえると後で見やすいので，TABを用いて整然と書く習慣をつけてください．

また，数の表記については16進数以外にも10進数やマイナスの数，そして加減算を含んだ式の状態で書くこともできます．

これらは，アセンブラがアセンブルするときに，自動的に16進数に変換してくれますが，具体的な例については本書での使用例を見て確認することにしましょう．

このMOV命令というのはいちばん多く使われる命令です．以下にその例を示します．

1. あるレジスタの値を別のレジスタに移す．移す側の値は変わらない．

```
MOV  AX, DX   ; AXにDXの値を代入する
MOV  SI, BX   ; SIにBXの値を代入する
MOV  DS, AX   ; DSにAXの値を代入する
```

2. 指示されたアドレスにはいっている値を各レジスタに代入する．アドレスの中身は変化しない．

```
DS:                    ；次の命令のセグメントベースをDSとする
MOV  AX,［0B300H］     ；DS:B300H番地にある値をAXに代入する
ES:                    ；次の命令のセグメントベースをESとする
MOV  BX,［SI］         ；ES:SI番地にある値をBXに代入する
```

＊［］で囲むと，その番地のなかにある値を意味する．このときのセグメントアドレスにはDSやESに格納されている数値が参照される．

3. レジスタの値を指示された番地のなかに移す．レジスタの値は変わらない．

```
DS:                    ；次の命令のセグメントベースをDSとする
MOV  ［0B300H］，AX    ；DS:B300H番地にAXの値を代入する
ES:                    ；次の命令のセグメントベースをESとする
MOV  ［SI］，BX        ；ES:SI番地にBXの値を代入する
CS:                    ；次の命令のセグメントベースをCSとする
MOV  ［0D500H］，AL    ；CS:D500H番地にALの値を代入する
```

以上がMOV命令の主な使用方法です．ここで，特筆すべきは，メモリ・アクセス時のセグメントアドレスの参照でしょう．実は，メモリ参照時のMOV命令を実行する前に，いちいちセグメントベースを決める命令(セグメント・オーバーライド・プリフィックス)を置いていましたが，〝DS：〟に限っては省略できるのです[*1]．

もっとも，使っているアセンブラがMASMであれば自動的にコードを省いてくれますから問題ありませんが……．

要するに，MOV命令とは数値を移動するための命令であると思えばいいのです．

覚えなければならない命令を書いていくと，それこそキリがありませんから，命令についての説明はこれが最初で最後です．この先，プログラムでわからない命令がある場合は，最初に紹介した参考書等を見て各自で調べてください．

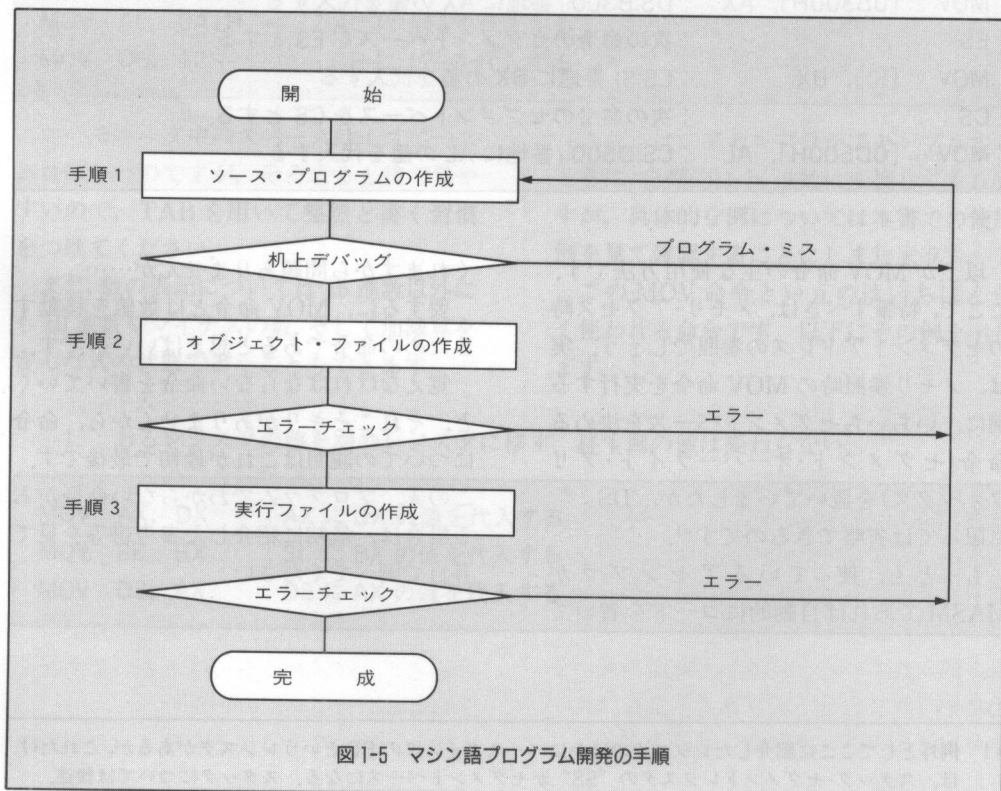
＊1　例外としてここに紹介したレジスタのほかにベースポインタのBPというレジスタがあるが，これだけは，スタック・セグメントレジスタの〝SS〟がセグメントベースになる．スタックについては後述．

6. プログラム … その作成と実行

これから，実際にマシン語のプログラミングをしていきますが，重要なことは，かならずテストの実行をするということです．そして，なるべく自分の手でプログラムを入力してください．これが，マシン語の書き方や命令，それにアセンブラの用法を覚える一番いい方法だからです．

さて，マシン語プログラムの開発には，基本的に，図 1-5 の 3 つの手順が行われます．

まず手順 1 のソース・プログラムの作成ですが，これは人間にわかりやすいニーモニックで記述したプログラムをテキストファイルとして作成することです．一般に，ソース・プログラムを作成するときには，テキスト・エディタを使います．MS-DOS にも EDLIN（エドリン）というエディタが付いてきますが，使い勝手があまりよくありません．この機会にスクリーンエディット（画面にファイル内容を表示し，任意の位置

図 1-5　マシン語プログラム開発の手順

にカーソルを移動して編集すること)が可能なエディタを1つ用意することをお勧めします.

手順2では,手順1で作成したプログラム・ファイルをMASMなどのアセンブラを使って,中間コードを含むマシン語コードに変換するのです.これを**オブジェクト・ファイル**といいます.なお,この変換する前のプログラム・ファイルは**ソース・ファイル**と呼ばれます.

最後の手順3では,手順2で作成されたオブジェクト・ファイルをリンカにより実行形式のファイルに変換します.

では,エディタを使ってp.36の**リスト1-1**を作成してください.プログラムの編集が終わったら,かならずセーブしておきましょう.このときのファイル名は〝LIST1-1.ASM〟とします.本書では,プログラムリストの最初に,リスト番号,タイトルに続けてつけるべきファイル名を表示しておきますので,かならずそのファイ名でセーブしてください.拡張子が〝.ASM〟となっているのはMASMではソース・ファイルの拡張子が〝.ASM〟であれば省略できるので便利だからです.

また,リスト1-1では,MASMに対する命令で疑似命令といわれる命令が記述されていますが,これについてはMASMのマニュアルを参照してください.

次に,MASMを使ってこのソース・プログラムをアセンブルしてみましょう.

```
A> MASM   LIST1-1 ; ⏎
```

アセンブル・リストとクロス・リファレンス・リストは今の段階では必要ありませんので,オブジェクト・ファイル名の後ろにセミコロン(;)を付けてあります.

この段階でアセンブル・エラーが生じれば手順の1に戻ってプログラムの編集をやり直してください.

次に,リンカを使って実行形式のファイルを作成します.

```
A> LINK   LIST1-1 ; ⏎
```

これでやっと,実行形式のファイル〝LIST1-1.EXE〟が完成します.さっそくこのプログラムを実行させてみましょう.

```
A> LIST1-1 ⏎
```

さて,何らかの画面の変化が生じたはずですが,気がつかれたでしょうか.画面左上にほんの数ミリの白い線が描かれているはずです.これが,すべてのグラフィックスの基礎で,ドットに直すと8ドットに相当します.

これは,図1-6のようにVRAMは,画面を8ビットつまり1バイト単位で管理しているからです.アドレス1つで8ドット分の表示を受け持っているというわけです.これらの1ドットを2進数の1桁つまり,1ビットと見立てると,通常のメモリと同じイメージで捉えられます.

それでは,プログラムが,どのように実行されているのか,1つ1つ確認してみましょう.まず図1-7を見てください.

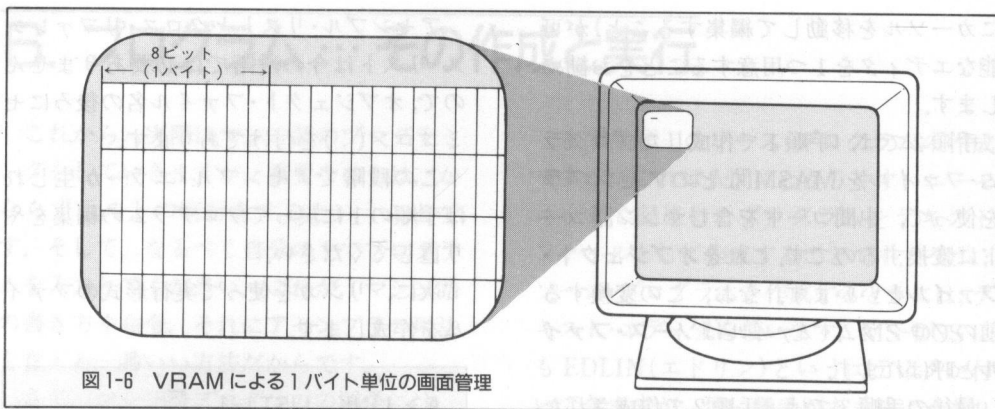



図1-6 VRAMによる1バイト単位の画面管理

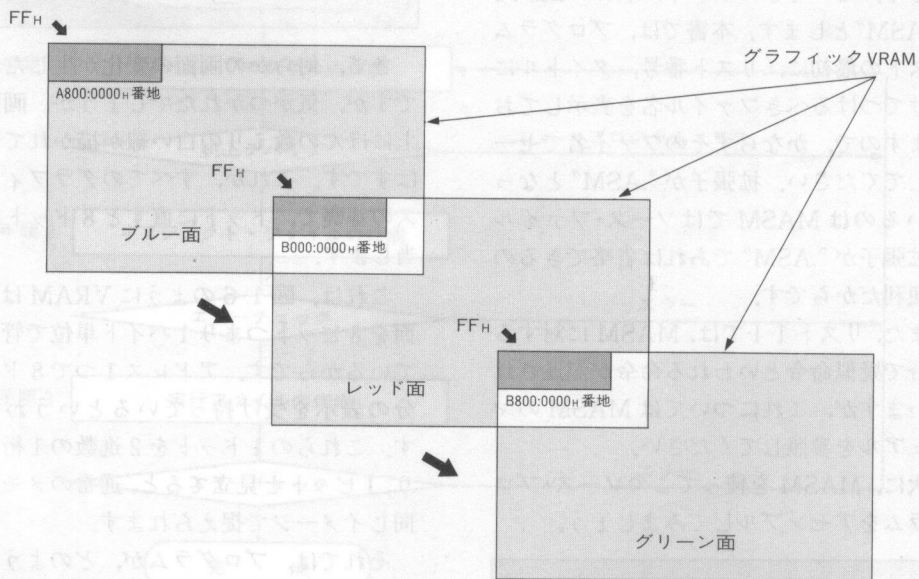



図1-7 リスト1-1の実行過程

PC-9801シリーズのVRAMは，主記憶空間の物理アドレスで言うと，A8000$_\mathrm{H}$～BFFFF$_\mathrm{H}$番地とE0000$_\mathrm{H}$～E7FFF$_\mathrm{H}$番地に割り付けられています．しかも，同じアドレス上に，選択可能なもう1組のVRAMが用意されています．これらの使い分けはポートA4$_\mathrm{H}$，A6$_\mathrm{H}$を介して行われます．0をA4$_\mathrm{H}$に出せば表画面が表示され，1であれば裏画面が表示されます．A6$_\mathrm{H}$は表と裏のどちらのメモリにアクセスするかを選択するためのポートです．0が表，1が裏となっています．

A8000$_\mathrm{H}$番地からはブルー面，レッド面，グリーン面が8000$_\mathrm{H}$バイトずつ順番に対応しています．また，E0000$_\mathrm{H}$番地から8000$_\mathrm{H}$バイトは輝度用のVRAMです．輝度用も含めて各色のVRAMの開始アドレスは画面左上に対応しています．

また，各色の面の開始アドレスは，セグメント・アドレスの設定しだいで，オフセット・アドレスがちょうど0になるようにとることができます．この場合の各開始アドレスの表記は次のようになります．

ブルー面　……　A800：0000$_\mathrm{H}$
レッド面　……　B000：0000$_\mathrm{H}$
グリーン面　……　B800：0000$_\mathrm{H}$
輝度用面　……　E000：0000$_\mathrm{H}$

このようにVRAMにアクセスする場合にはオフセット・アドレスが0から始まるようにセグメント・アドレスを設定するのが普通ですが，これは，画面上の同じ場所に色々な色を出す場合，セグメント・アドレスを変えるだけですみ，便利だからです．

プログラミング・テクニック以前のテクニックとして，通常はこのようにセグメント・アドレスを設定すると，覚えておいてください．もちろん，セグメント・アドレスとオフセット・アドレスの組み合わせ方は，このほかにもたくさんあります．

さて，各オフセット・アドレスとCRT画面との関係は，**表1-1**のようになります．

これで，マシン語プログラムのためのウォーミング・アップはOKです．2章ではリスト1-1をより発展させたキャラクタ・

座標	0	128	256	384	512	639
0	0000H	0010H	0020H	0030H	0040H	004FH
1	0050H	0060H	0070H	0080H	0090H	009FH
2	00A0H	00B0H	00C0H	00D0H	00E0H	00EFH
⋮	⋮	⋮	⋮	⋮	⋮	⋮
397	7C10H	7C20H	7C30H	7C40H	7C50H	7C5FH
398	7C60H	7C70H	7C80H	7C90H	7CA0H	7CAFH
399	7CB0H	7CC0H	7CD0H	7CE0H	7CF0H	7CFFH

表1-1　VRAMとグラフィック座標の関係

パターンの表示から移動へと進めていきます．といっても，基本はこのリスト1-1ですから，リスト1-1さえしっかり理解できれば，山も半分ぐらい登ったようなものです．なお，リスト1-1の最後には，スタックエリアが設定されていますが，スタックに関しては後述します．

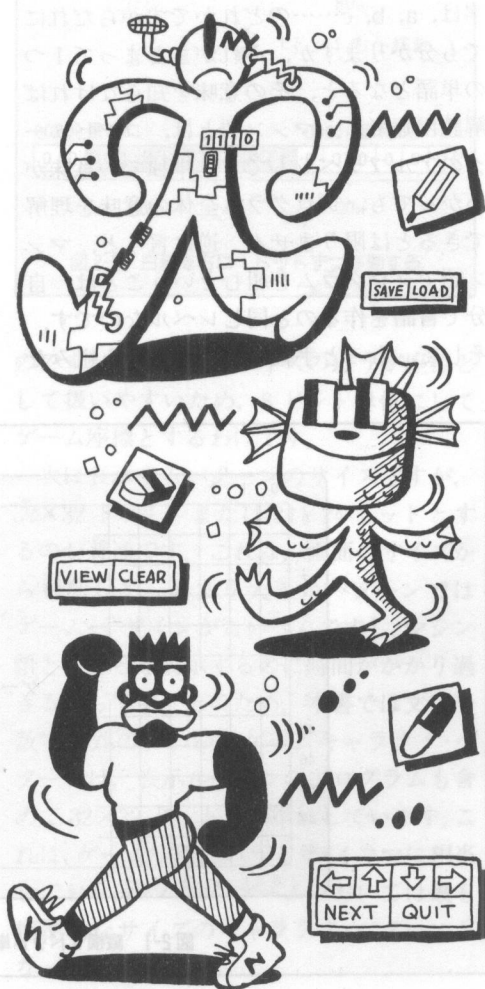
リスト1-1　線を引く(LIST 1-1.ASM)

```
;******* LIST 1-1 *******

BLUE    equu    ØA8ØØH                  ブルー面セグメント値
RED     equ     ØBØØØH                  レッド面セグメント値
GREEN   equ     ØB8ØØH                  グリーン面セグメント値

CODE    segment                         命令の置かれているセグメントの始まり

        assume  CS:CODE,SS:STSEG

        CALL    GINIT                   グラフィック・システムの初期化
        MOV     AX,BLUE                 データ・セグメント値の設定(ブルー面)
        MOV     DS,AX                   AXレジスタを介してデータ・セグメントをセット
        MOV     DL,ØFFH                 DL←FFH
        MOV     DI,Ø                    DI←0 …… VRAMアドレス
        MOV     [DI],DL                 VRAMのDS:[DI]にFFHを格納
        MOV     AX,RED                  データ・セグメント値の設定(レッド面)
        MOV     DS,AX                   AXレジスタを介してデータ・セグメントをセット
        MOV     [DI],DL                 VRAMのDS:[DI]にFFHを格納
        MOV     AX,GREEN                データ・セグメント値の設定(グリーン面)
        MOV     DS,AX                   AXレジスタを介してデータ・セグメントをセット
        MOV     [DI],DL                 VRAMのDS:[DI]にFFHを格納
        MOV     AH,4CH                  ファンクションコールのパラメータをセット
        INT     21H                     ファンクションコール(MS-DOSシステムへ戻る)

GINIT:  ;Graphic system INITialize
        MOV     AX,4ØØØH                グラフィック画面の表示開始コマンドをセット
        INT     18H                     ROM内ルーチン・コール
        MOV     AX,42ØØH                グラフィック画面モード設定コマンドをセットする
        MOV     CH,ØCØH                 640×400ドット・カラー・モードで初期化とする
        INT     18H                     ROM内ルーチン・コール
        RET                             リターン

CODE    ends                            CODEと名付けたセグメントの終了

STSEG   segment stack                   スタック用セグメントの開始
        db      1ØØH dup (Ø)            100Hバイト確保
STSEG   ends                            スタックセグメントの終わり

        end                             プログラム・エンド
```

2章
●キャラクタ・パターンの表示と移動

●マシン語ゲームのすばらしさは，何と言っても画面の中を高速に動き回るキャラクタです．これぱっかりは，BASICではそう簡単に実現できません．マシン語をマスターしたい一番の理由も，たいていの場合は，こんなところにあるのではないでしょうか．

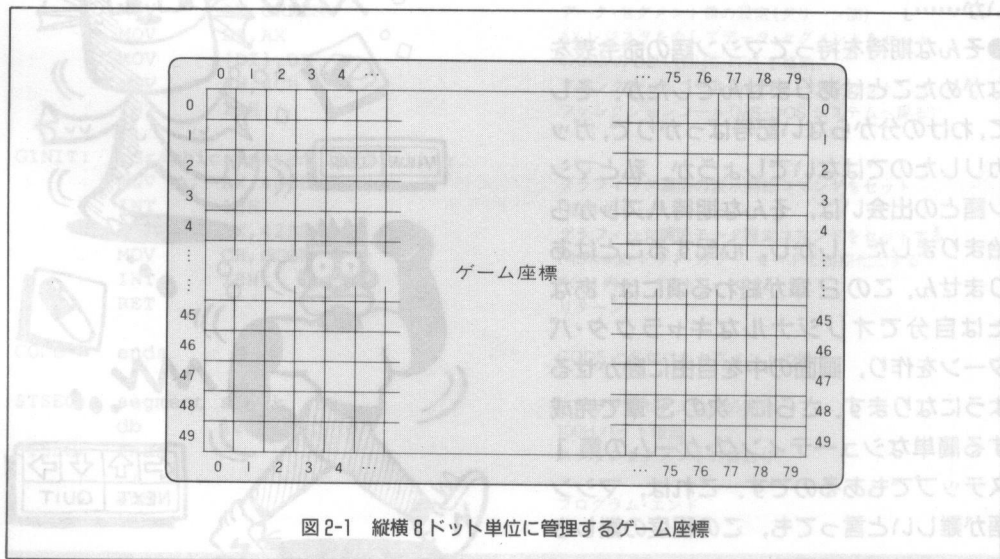
●「マシン語を使えば，キャラクタを思い通り動かすことができる．きっと，マシン語にはBASICにはないキャラクタ表示命令とか，それを動かす命令があるのではないか……」

●そんな期待を持ってマシン語の命令表をながめたことはありませんでしたか．そして，わけの分からない記号ばっかりで，ガッカリしたのではないでしょうか．私とマシン語との出会いは，そんな期待ハズレから始まりました．しかし，心配することはありません．この２章が終わる頃には，あなたは自分でオリジナルなキャラクタ・パターンを作り，画面の中を自由に動かせるようになります．さらに，次の３章で完成する簡単なシューティング・ゲームの第１ステップでもあるのです．これは，マシン語が難しいと言っても，この程度の難しさだという証明なのです．

1. 座標…ゲームのためのゲーム座標

アルファベットはわずか26文字しかありませんが，だから英語がやさしいと考える人はまずいません．それは，言語というものが文字を組み合わせて作られている，ということを知っているからにほかなりません．どんなに難しい単語でも，1文字1文字は，a, b, c……のどれかですからだれにでも分かりますが，それがまとまって1つの単語となると，その意味を知らなければ解読困難です．マシン語とは，コンピュータのアルファベットです．単独での意味がわかっても，プログラム全体の意味を理解できるとは限りません．逆に言うと，マシン語でプログラムを組むということは，自分で言語を作るのと同じレベルなのです．そして，もっと手軽にプログラムを組みたい人のために用意されているのが，BASICであるといえるのです！

何だか，スゴク難しいことをやろうとしているように思えるかもしれませんが，BASICのように使用目的がハッキリしていない言語をマシン語で作ろう，というのではありません．これから作るマシン語プログラムは，自分のゲームにだけ通用し，しかも使用上の制限は勝手に付けていいのですから，いたって気楽なものです．

この使用目的を限定するということが，結局は処理速度を速くできることにつながっていくわけです．そこで，まずはゲームの顔とも言えるグラフィック画面に対して，ゲームに便利なように制限を付けることにします．

図2-1　縦横8ドット単位に管理するゲーム座標

BASICの場合には，グラフィック画面を(0,0)-(639,199)または，(0,0)-(639,399)という，どちらかの座標系を1ドット単位で管理していました．

どちらの座標系を用いるかは，CRTや，プログラムに依存するわけですが，本書では，(0,0)-(639,399)の座標系を使っていきます．そのほうがより繊細なグラフィックスが楽しめるからです．

また，パターンの移動座標は，ドット単位ではなく，横8ドット，縦8ドットの正方形を1マスとして，管理します．これを本書では「ゲーム座標」といい，座標で表すと(0,0)-(79,49)ということになります．

図2-1を見てください．ゲーム・デザインを考えるときにもこの座標を基準にして作成することになりますので，最初に用意した方眼紙にこの座標を書き込んでおくと便利です．なぜ，1ドット単位で管理しないのかという最大の理由は，処理に時間がかかり過ぎるということです．1章で，グラフィック画面に短い白線を描きましたが，これを右に1ドットだけずらすとなると，図2-2のように，データを入れるVRAMのアドレスは2バイトにわたってしまい，データも2つ用意しなければなりません．また次にもう1ドット右にずらすとなると，また，別の2つのデータが必要になります．

もっとも，最近では，EGC(Enhanced Graphic Charger)という強力な機能が搭載されていますから，横方向でもドットごとの管理が楽になりましたが，機種が限られるため，ここでは横8ドット単位とします．縦は別に1ドット単位でも問題はないのですが，縦横同じサイズのほうが座標として扱いやすいため，8ドット単位にしてゲーム座標とするわけです．

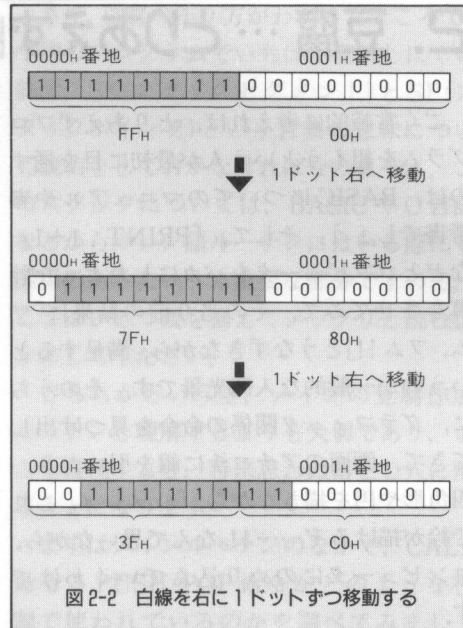



図2-2　白線を右に1ドットずつ移動する

次に表示するパターンのサイズですが，32×32ドット，または24×24ドットとするのが普通です．これは，画面のサイズから判断して，あまり大きいパターンではゲーム・デザインがたいへんですし，マシン語といえども表示するのに時間がかかり過ぎるからです．そのため，本書では文字や数字以外のいわゆるゲームキャラクタ・パターンは，表示ルーチンのプログラムも含めて32×32ドットを基準にしています．これは，ゲーム座標でいうと4×4コマに相当しており，パソコン・ゲームにおいては最も標準的なサイズのキャラクタ・パターンとなっています．

2. 豆腐…とりあえず白い四角形を表示

ごく常識的に考えれば，とりあえずプログラムを組もうという人が最初に目を通すのは，BASICについてのマニュアルや参考書でしょう．そして，『PRINT 1+1』などとコンピュータをバカにしたような計算をさせてみて，その当り前の結果に「フム，フム!!」とうなずきながら，満足するというのが一般的な入門光景です．そのうちに，グラフィック関係の命令を見つけ出してきて，画面のアチコチに線を引いたり，四角形や円を描きながら，「シメシメ，これで絵が描けるぞ……!!」なんて思いながら，コンピュータにのめり込んでいくわけです．

マシン語を覚える際にも，このように視覚に訴えながら進んでいくと，理解する楽しさが増してきます．プログラムを追うだけでは，どうしても面白さ，わかりやすさという点で不満が残ってしまうものです．ちょうど，小説よりもマンガのほうが，情景がハッキリするのと同じことです．

頭の中だけで理解するより，視覚に訴えて理解するほうが間違いも少ないし，進歩の度合も速いといえるでしょう．

それでは，任意の座標位置に32×32ドットの白い正方形を表示するプログラムを作成してみましょう．

リスト2-1を見てください．ラベルというものはプログラムを作った本人以外には，なかなか理解しにくいものなので，省略前のものも載せてあります．ただし，英文法は無視していますので，そのつもりで見てください．本文では命令そのものについての説明は避け，重要な語句やプログラムの概略を中心に説明をしてあります．では，まず白い真四角な豆腐ができるまでの工程を示した**図2-3**を見てください．

このプログラムのメイン部分はCODEと名付けたセグメント内(segment～ends で囲まれた部分)のプログラムです．これは，大きく分けると5つのブロックからできています．

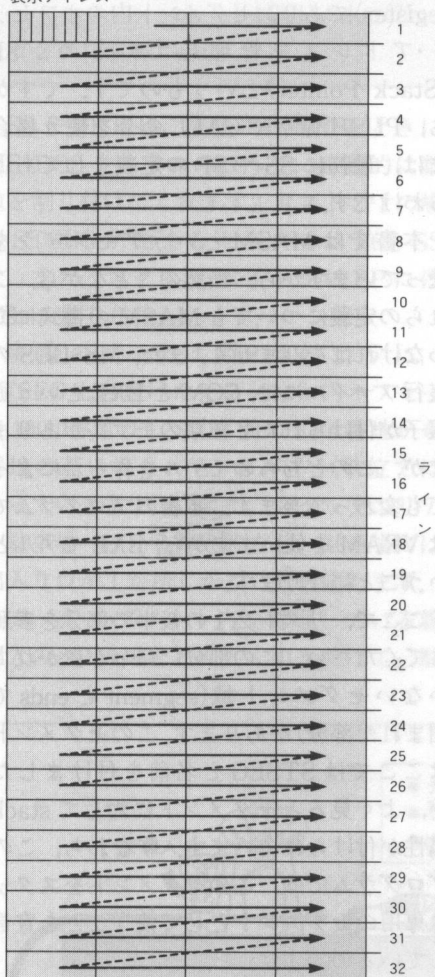
1～5行はVRAMのセグメント値などに名前を付けています．15～21行のGINITルーチンでは，グラフィックス・システムの初期化をPC-9801のROM内ルーチンを利用して行っています．

22～31行のTOUFUルーチンは，(BX, CX)で示される位置を後述するXYADRルーチンで実際に四角形を表示するアドレスに変換したのち，ブルー，レッド，グリーンの3画面に四角形を表示します．42～48行のXYADRルーチンは，BX, CXに格納されたゲーム座標の(X,Y)座標から，VRAMの実アドレスに変換してBXに入れています．8～14行のTESTルーチンは，BX, CX各レジスタにゲーム座標上の表示アドレスを入れてTOUFUルーチンをコールする部分です．なお，本書のプログラムは，すべてこのTEST(メインルーチン)から実行するようにしています．そこで，プログラムを読むときには，まずTESTルーチンから読み始めるとプログラム全体の構成が理解しやすいでしょう．

1 豆腐の表示アドレスを求める
2 ブルー面に32×32ドットの正方形を描く

表示アドレス

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32

ライン

(表示アドレスから──→に沿ってFF$_H$を入れていく)

3 レッド面に32×32ドットの正方形を描く
(2と同様)

4 グリーン面に32×32ドットの正方形を描く
(4と同様)

図2-3 豆腐ができるまで

さて，豆腐の作り方がわかったところで，今回のプログラムでいちばん理解しにくい部分，SS(スタックセグメント)とSP(スタックポインタ)という言葉の意味について説明をしておかなければなりません．このスタックについては，BASICやC言語などからマシン語ルーチンにはいる際にも関係のあるたいへん重要な部分ですので，ここはひとつ腰を据えてジックリと読むようにしてください．

どちらかというと，メインの豆腐作成ルーチンを理解するよりも大切であり，ここを軽視すると，将来思わぬ落とし穴に陥ることになります．

まずはメインルーチンのなかで，CALL命令とPUSH/POP命令がどのような役割で使われているのかを調べてみましょう．CALL～RET命令は，C言語で言うと関数に，BASICではGOSUB～RETURNに対応しています．これらに対して，PUSH/POP命令というのはBASICやC言語などにはない考え方で，一時的にレジスタの値を保存しておき，必要なときに取り出すという命令です．保存するときの命令がPUSH，取り出す命令がPOPです．これは，BASICやC言語と違い，変数を自由に作れないマシン語では，非常に便利な存在となっています．

次に，CPUの動きに目を向けてみます．CPUは，CS(Code Segment register)とインストラクション・ポインタ(Instruction Pointer)によって参照されるアドレスから命令を取り出してきます．このときインストラクション・ポインタには次の命令や，パラメータの格納されているアドレスのオフ

セットが格納されています.

命令の実行が終わると，再びCSとインストラクション・ポインタによって参照される命令を読み，また次のアドレスのオフセットをインストラクション・ポインタに記録する……，ということを繰り返し行っているのです．結構，手間のかかることをしていますね．しかし，これだけではRET命令に出会っても，もとの流れに戻ることはできません．きちんと戻るためには，CALL命令があった場所で，CALL命令の次の命令があるアドレスを，インストラクション・ポインタとは別のどこかに記録しておかなければならないはずです．このとき，同じセグメント内であればそのオフセット・アドレスを，また，別なセグメントであれば，戻るべきセグメント・アドレスまでも記録します．

ところで，PUSH命令は一時的にレジスタの値を保存するといいますが，いったいどこに保存しているのでしょうか．CALL命令にしてもPUSH命令にしても，プログラムによって使用される回数が違うので，そのための記録領域をどのくらい用意すればいいのかまったく不明です．そこで，これらのデータを記録するために，メモリの一部を最初に記録専用の領域として用意し，そこにレジスタの値を保存していきます．

このような特殊な記録領域を「**スタックエリア**」といいます．そして，このスタックエリアを管理する特別なレジスタが2つ用意されているのです．

スタックエリア用のセグメントを管理しているのがSSレジスタ(Stack Segment register)であり，セグメント内のオフセット・アドレスを管理しているのがSP(Stack Pointer)というものです．ですから，PUSH命令やCALL命令を使う場合には，最初にSSやSPの定義をしてあげるわけです．

本書ではMASMというアセンブラを使っていますから，当然のことながら，これらの定義についてもMASMの書式に従わなければなりません．また，MS-DOSの実行ファイルには，COMとEXEという拡張子が付けられた2種類のモデルがありますが，このどちらのモデルを作るかによっても変わってきます．本書のプログラムではVRAMを使いますから，EXEモデルということになります．

ここで，リスト2-1の後半の部分を参照してください．この部分に命令が置かれていないセグメント域(segmentとendsで囲まれた部分)があります．このセグメントはここではSTSEGと名前を付けましたが，よく見るとセグメントに対してstack属性が付けられています．すなわち，このプログラムでは，このセグメントをスタック専用のセグメントとして使うことを宣言しているのです．

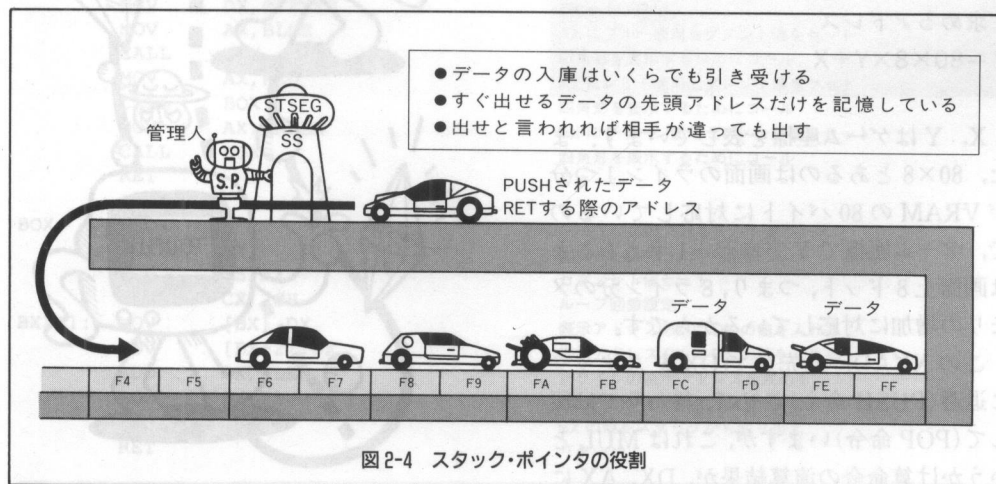
スタックエリアの大きさは自由に設定できますが，本書ではこの大きさを100Hバイトに定義しています．

スタック・セグメントの定義
セグメント名　segment　stack　……　スタック属性のセグメントの宣言
db　100H　dup(?)　……　100Hバイトの領域を確保する
セグメント名　ends　……　セグメントの終わり

SSやSPには，実際に作動するときにMS-DOSシステムのほうで，自動的に数値を割り付けてくれます．リスト2-1の場合STSEGが始まるセグメント値をSSに，また，スタック・ポインタのSPには100Hが割り付けられます．

実際のデータは図2-4のように，STSEGと名付けたセグメント内オフセット100H番地の1バイト前のFFH番地から番地の若いほうへと，2バイト単位ではいっていきます．ここで，スタックエリアを駐車場へはいる車とみなし，駐車場には出入り口が1か所しかないと仮定します．スタック・セグメント内を管理している管理人のSPは車をどんどん引き受けて，駐車場の一番奥から入れていきます．しかし，車を出すときは出口に近いものからしか出ませんから，SPは出口に一番近い車の駐車位置(番地)だけをつねに覚えているのです．このように最後に入れたものを最初に出すというルールを，LAST IN FIRST OUT，または，FIRST IN LAST OUTの原則といいます．

また，この管理人のSPはたいへんいい加減で，車を出しに来た者には，もとの預け主でなくても車を渡してしまうのです．たとえば，AXから預かった車でも，DSが取りにくればDSに，BXが取りにくればBXにという具合いに，いちいち確認などせずに渡してしまうのです．ですから，駐

図2-4　スタック・ポインタの役割

車場のオーナー(結局はプログラムを組むあなたのことですよ)が出し入れの順番を，シッカリ把握しなければならないのです．

この原則を踏まえた上で，CALL命令やPUSH/POP命令を使わないと，恐ろしい暴走に出会うことになります．しかし，実際には1つのCALLルーチンのなかで，PUSH命令とPOP命令の使用回数が同じであって，スタックエリアとして100Hバイトぐらいのメモリを確保してあれば，とくに問題は起きないものです．

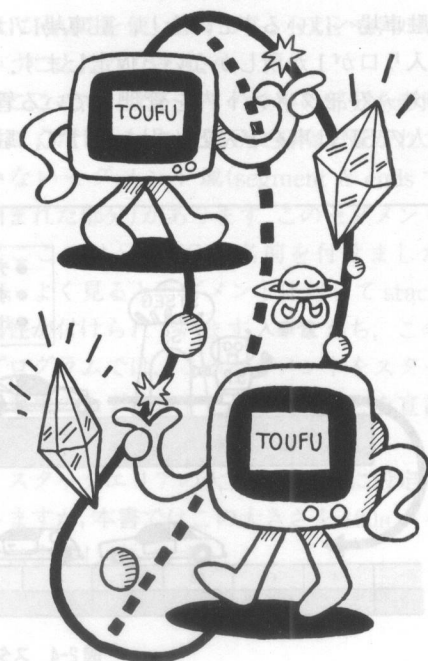
スタックを操作する〝RET N〟(Nは任意の偶数)などの命令のように不要なスタック・データを破棄する命令もありますが，原則的にPUSH/POP命令の数は合わせると覚えておいてください．

さて，このプログラムのなかでとくに説明を要するのは，ゲーム座標から実際のVRAMのアドレスに変換しているXYADRという部分ですが，ここでの計算式は次のとおりです．

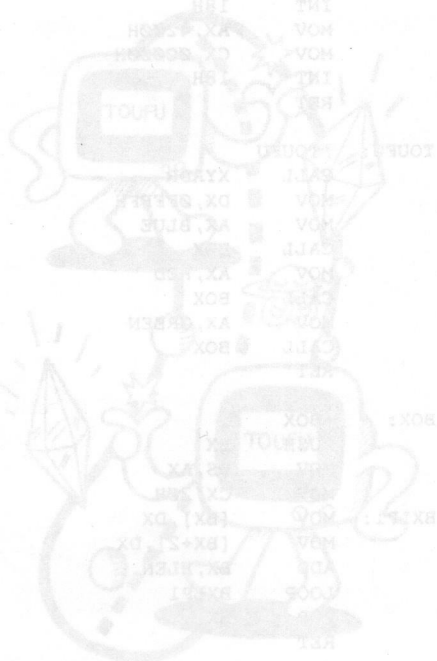
求めるアドレス
＝80×8×Y＋X

X，Yはゲーム座標を表しています．また，80×8とあるのは画面のライン1つ分がVRAMの80バイトに対応しているので，ゲーム座標でY方向が＋1増えることは画面上8ドット，つまり，8ライン分のメモリの増加に対応しているからです．

このXYADRの先頭でDXをスタックに退避(PUSH命令)させて，終わりでは戻して(POP命令)いますが，これはMULというかけ算命令の演算結果が，DX，AXにそれぞれ上位，下位の順に格納されるからです．もちろん演算結果の最大値がVRAMの表示アドレスの最大値の7CFFHを越えることはありませんので，DXの値はつねに0となり，プログラム上は，必要ないことになります．このように不必要にレジスタを使用する場合には，思わぬバグにつながりますから，DXの値を保存したというわけです．

では，テストの実行です．MS-DOSのコマンド・ラインから〝LIST2-1⏎〟と入力してください．アッと言う間に「豆腐のイッチョ上がり」となりましたね．

リスト2-1 豆腐の表示(LIST2-1.ASM)

```
;******** LIST 2-1 *********

HLEN     equ      8Ø                  1ライン分のVRAMアドレスの増分
HLEN4    equ      8*8Ø                8ライン分のVRAMアドレスの増分
BLUE     equ      ØA8ØØH              ブルー面セグメント値
RED      equ      ØBØØØH              レッド面セグメント値
GREEN    equ      ØB8ØØH              グリーン面セグメント値

CODE     segment                      命令の置かれているセグメントの始まり

         assume CS:CODE,SS:STSEG

PTEST:   ;Program TEST                ――メインルーチン
         CALL     GINIT               VRAMの初期化
         MOV      BX,Ø                X座標0
         MOV      CX,Ø                Y座標0
         CALL     TOUFU               (BX,CX)に豆腐を表示
         MOV      AX,Ø4CØØH           コマンド番号セット
         INT      21H                 MS-DOSへ

GINIT:
         MOV      AX,4ØØØH            グラフィック画面表示開始コマンド・セット
         INT      18H                 ROM内ルーチン・コール
         MOV      AX,42ØØH            グラフィック画面モード設定コマンド・セット
         MOV      CX,ØCØØØH           640×400ドット・カラー,表示バンク0をセレクト
         INT      18H                 ROM内ルーチン・コール
         RET                          リターン

TOUFU:   ;TOUFU                       ――四角形を表示するルーチン
         CALL     XYADR               (BX,CX)からBXに表示アドレスを求める
         MOV      DX,ØFFFFH           DX←FFFFH
         MOV      AX,BLUE             AXにブルー面用セグメント値をセット
         CALL     BOX                 四角形を表示するためにコール
         MOV      AX,RED              AXにレッド面用セグメント値をセット
         CALL     BOX                 四角形を表示するためにコール
         MOV      AX,GREEN            AXにグリーン面用セグメント値をセット
         CALL     BOX                 四角形を表示するためにコール
         RET                          リターン

BOX:     ;BOX                         ――1つの四角形を表示するルーチン
         PUSH     BX                  BXの値をスタックへ退避
         MOV      DS,AX               セグメント値セット
         MOV      CX,2ØH              ループ回数設定
BXLP1:   MOV      [BX],DX             表示アドレスBXにDXの値を入れる
         MOV      [BX+2],DX           表示アドレスBXにDXの値を入れる
         ADD      BX,HLEN             次ラインの表示アドレスを求める
         LOOP     BXLP1               CX←CX−1として,CX=0までループする
         POP      BX                  BXの値をスタックから取り出す
         RET                          リターン
```

```
XYADR:  ;XY  to  ADdRess          ―― 表示アドレスを求めるルーチン
        PUSH     DX               DXの値を退避
        MOV      AX,HLEN4         AX 被乗数セット
        MUL      CX               DX：AX ← CX×AX
        ADD      BX,AX            BX ← BX＋AX
        POP      DX               DXの値をスタックから取り出す
        RET                       リターン

CODE    ends                      CODEと名付けたセグメントの終わり

STSEG   segment  STACK            スタックセグメントの宣言
        db       1ØØH dup (?)     メモリ領域を100Hバイト確保
STSEG   ends                      スタックセグメントの終わり

        end                       プログラム・エンド
```

3. パターン … キャラクタの作成

豆腐作りの修行はいかがでしたか．画面のなかの何処にでも豆腐を作れるようになれば，もう一流の豆腐職人です．ところで，このルーチンの先頭に付いているTOUFUというラベル名，これを1つの言葉と考えてみるとどうでしょうか．これは，すでにアルファベット的な文字ではなく，オリジナルな言語であるといえます．それが証拠にBASICやC言語にはTOUFUなどという命令は，どこを捜してもありません．サイズは一定，表示位置はゲーム座標による，というような制限はありますが，そういう言葉なのですからそれでいいのです．自分で作った，自分のためだけの言葉ですから，他人が使うことなど考える必要もないのです．こんなところが，マシン語のたまらない魅力であり，いっぽう，とっつきにくくしていた原因でもあったのです．しかし，豆腐一丁でその壁はもろくも崩れたことでしょう．「豆腐の角に頭をぶつけて，死んでしまえ!!」というのは，この壁に対する格言だったのですネ？

これから，豆腐を卒業して実際にパターンを画面に表示する段階にはいるわけですが，パターン・データを表示するには，まずそのデータがなければなりません．ここでは，パターン・データ作成の方法として，パターン・エディタを実際に使いながら，次節で使うデータを作成することにしましょう．

まずデータの作成方法ですが，方眼紙にドットで絵を描いて，それを手作業で16進数に直す……，なんていうことを考えた方はいないと思いますが，現実はそれと同じことをコンピュータにさせて作るのです．

Appendix 2のMSPTERがそのためのプログラムですが，ここでテスト的に利用するだけでなく，将来も使えるように色々と便利な機能を付けてあります．マシン語の勉強とは少し離れるかもしれませんが，これなくしてはパターン表示のテストもできませんから，頑張って打ち込んでください．

リストを打ち終えたらデータをセーブするのは，もう常識ですね．モデルはEXE型となりますので，SYMDEB(MS-DOSのデバッガ)などで打ち込んだ場合には，一度ディスクにセーブしたあと，MS-DOSのREN命令で拡張子をEXE型に変更してください．

では，走らせてみましょう．プロンプトの表示が次ページの図2-5のように変わります．

この状態では，主にファイル関係のコマンドが使えるようになっています．[L]はファイルのロード，[S]はエディットしたデータのセーブ，[D]はDIRコマンド，[A]～[C]はドライブチェンジとなっています．[E]を押すと，エディット画面になります．

エディット画面の右側中ごろにはカラーパターンが並んでいますが，そのなかに四角い枠があります．これは，透明(背景となる)を意味するものです．ただし，透明の

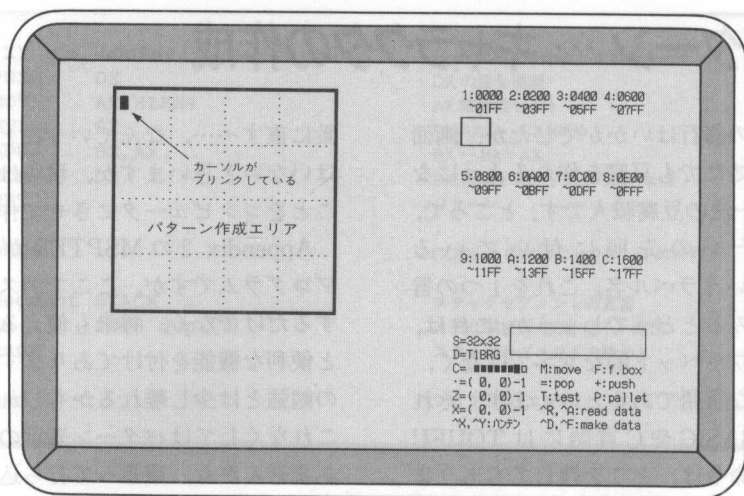



図 2-5　パターン・エディタの画面

データがあっても，重ね合わせ表示用のプログラムがないと何の役にも立ちませんので，6章までは関係のない色(?)といえます．なお，透明色を表す四角い枠の色は，8色のなかから選べます．

では，さっそく巻頭口絵4のキャラクタ・パターンのなかから，好きなものを1つ作成してみましょう．キャラクタのサイズはSコマンドで32×32にしてください．

なお，パターン・エディタの機能は表2-1に掲載してあります．

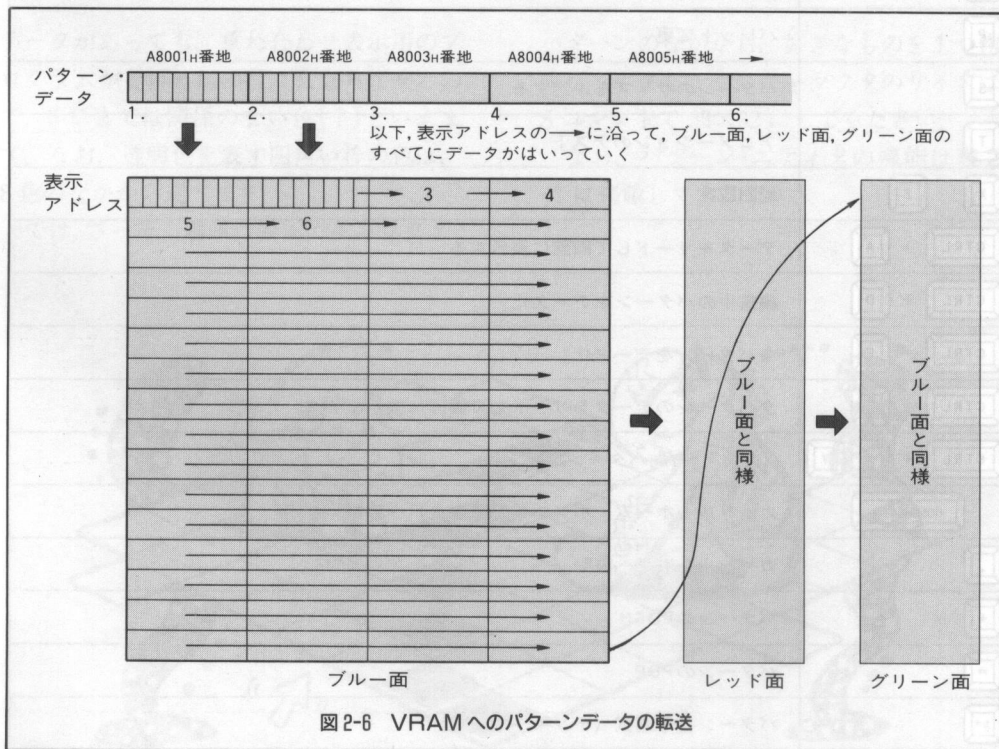
キー	機能
ESC	キャラクター・パターンのエディット状態からMSPTERのコマンド受付状態へと制御を移す
C	透明色を含めて9種類の中から色を選択する
D	透明色のタイプ決定 BRG：透明色無し，T1BRG：透明ビット＝1のデータ，T0BRG：透明ビット＝0のデータ
F	指定範囲を選択されている色で埋める
M	指定範囲を移動（コピー）
P	パレット変更
S	パターンのサイズを決定する
T	アニメーションのテスト
X ，Z	範囲設定
CTRL ＋ A	データをリードして画面に表示する
CTRL ＋ D	編集中のパターンをデータ化
CTRL ＋ F	全パターンをデータ化
CTRL ＋ R	全パターンのデータをリードして画面に表示する
CTRL ＋ X Y	X方向，Y方向パターン反転
HOME／CLR	カーソルをホーム・ポジション（左上）へ移動
.	カーソルを右下へ移動
＋	パターンのPUSH
＝	パターンのPOP
カナ	パターン番号選択（テンキーで決定）

表2-1　パターン・エディタの機能表

4. パターン表示…キャラクタ登場

1つのゲームを作るときに，パターン数はいったいいくつぐらい必要になるかというと，これが千差万別なのです．文字や数字のパターンを除くと，少ないものでは20～30程度，場合によっては1000以上のパターンを使用しているものもあります．パターン数を多くすれば，それだけ動きがなめらかになりますが，メモリの効率や作る労力のことを考えると，一概に多ければいいともいえません．それよりも，少ないパターン・データで多く見せるということのほうが，必要なのかもしれません．いずれにしても，ゲームを作るということは，パターン作成という地味な作業も避けては通れない部分だということです．

テスト用とはいえ，あなた自身で作ったパターン(キャラクタ)のデータが揃いました．これで本当に豆腐とオサラバできることになったわけです．しかし，パターンを表示するプログラム(**リスト 2-2**)は，これまでVRAMに入れていたFFHをパターンに置き換えるだけですから，リスト2-1

図2-6 VRAMへのパターンデータの転送

にパターンをロードするプログラムを付け加えるだけで，あとは，ほとんど変わりがないといえます．

データをVRAMに記憶する方法は，図2-6に示したように，基本は豆腐作りと同じ考え方です．パターン・エディタによるデータ作成時には，このデータの流れが逆になっていただけですから，描かれるパターンが同じになるのも当然のことですね．

リスト2-2で注目すべき点が1つあります．それはLOOP1のなかで次のような条件分岐をしていることです．

```
DEC   DX
JNE   LOOP1
```

これは一見すると，DXの値がゼロか否かを判定してLOOP1へジャンプしているように見えますが，結果的にはそうであっても，実際はDXの値を見ているのではなく，フラグレジスタ中にあるゼロフラグを見て判定しているのです．

では，フラグレジスタとは何かというと，実は数値を代入するこれまでのレジスタと違い，足し算や，引き算，論理演算，比較などの各種演算をした結果によって，ある決まった反応を示す特殊なレジスタなのです．このレジスタの内容を表2-2に示しておきます．

フラグレジスタの役目は，演算に対しビット単位で1か0を示すということです．そして，フラグの場合はビットが1になっている所を「フラグが立っている」とい

ビット	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
	予備	予備	予備	予備	Oフラグ	Dフラグ	Iフラグ	Tフラグ	Sフラグ	Zフラグ	予備	Aフラグ	予備	Pフラグ	予備	Cフラグ

フラグ	内容
Oフラグ	オーバーフローフラグ，演算結果が桁溢れをおこした場合セットされる
Dフラグ	ディレクションフラグ，ストリング命令のアドレスの増減を決定する
Iフラグ	インタラプトフラグ，割り込みの許可，禁止を表す
Tフラグ	トラップフラグ，シングルステップモードにする
Sフラグ	サインフラグ，演算結果上位ビットの値がそのままはいる
Zフラグ	ゼロフラグ，演算結果が0であれば1，0でなければ0となる
Aフラグ	補助キャリーフラグ，ビット3から，またはビット7のキャリーを表している
Pフラグ	パリティ・フラグ，演算の結果，ビットの1が偶数ならセットされる
Cフラグ	キャリー・フラグ，演算の結果のキャリーを表している

表2-2　フラグレジスタの内容

います．とくにゼロフラグの場合は，「ゼロのときには1になる」などと覚えようとすると混乱しますから，「ゼロになったらゼロフラグが立つ」と単純に覚えたほうがハッキリします．

これらのフラグのなかで，よく使われるのはゼロフラグとキャリーフラグの2つで，その他については徐々に意識すれば十分です．ここで無理に覚えなくても，マシン語に慣れてくれば自然と覚えてしまいます．

とりあえずこの段階では，以下の2つのことを覚えておいてください．

- ゼロになったらゼロフラグが立つ
- 演算の結果，桁上げ，桁借りがあったらキャリーフラグが立つ

さて，実際のフラグの利用ですが，さきほどの例のように条件分岐としてよく使用します（キャリーフラグは，算術，ローテート，シフト命令でも用いる）．

さきほどの例が，なぜDXの値を見てジャンプしているわけではないのか，次のようにすると明確になります．

```
DEC   DX      ；DX ← DX－1．この演算の結果はフラグに反映される
MOV   DX，0   ；DX ← 0とする．MOV命令ではフラグは変化しない
JNE   LOOP1   ；ゼロフラグが立っていなければLOOP1へジャンプする
```


プログラム的には，まったく意味がなくなってしまいますが，〝DEC　DX〟をした時点でDXの値が0でなければ，LOOP1にジャンプすることになります．このように，命令によってフラグは影響を受けたり受けなかったりしますので，条件分岐をするときには注意が必要です．

さきほど，作成したパターン・データを，プログラムの所定のアドレスにロードするプログラムでは，また別なフラグの使用例があります．

それは，MS-DOSのファンクション・コールを利用している部分です．〝INT 21H〟は実際にコールしている部分ですが，処理が正常に行われたかどうかの情報がさきほどのキャリーフラグに格納されて返ってくるのです．処理がエラーで返ってきたときには，キャリーフラグが立っていますから，ここで条件分岐によるエラー処理をするわけです．

このように，まったくメモリを使わずに，各サブルーチン間での情報のやりとりにフラグを利用する場合もあるのです．

さて，フラグが理解できたところで，このパターン表示プログラムのテストをしてみましょう．さきほど作成したパターンデータをロードする場合には，ロードファイル・テーブルに登録してあるファイル名がパターンエディタでセーブしたファイル名と一致していなければなりません．もし，ロードするファイルが存在しない場合には，エラーメッセージを表示するようにしてあります．

TESTルーチンのBX，CXの数値が

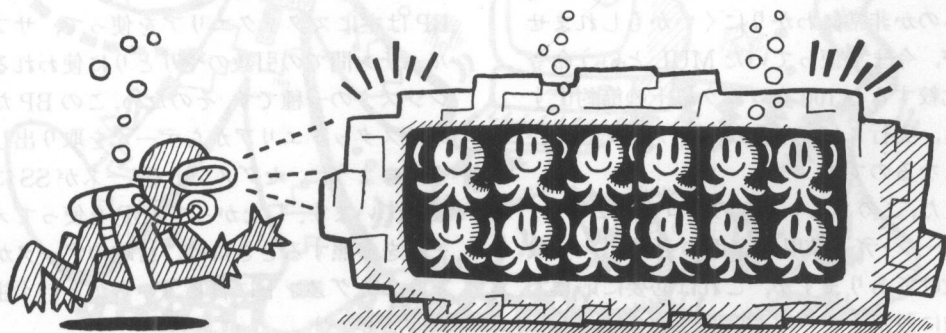
ゲーム座標のX，Yに対応していますから，これらのレジスタの値を変えることにより，画面の好みの位置にオリジナル・パターンを表示できるようになったはずですが，……いかがでしょうか？

このプログラムでも，簡単なゲームであればとくに問題はありませんが，まだまだ本格的なパターン表示ルーチンとはいえません．その理由は速度です．実際のマシン語ゲーム(アクションタイプ)では，7~8割がパターンを表示するための時間に費やされています．ですから，パターン表示ルーチンをできるだけ高速にすることが，すなわちゲームの高速化につながるのです．このことは，ゲーム中に表示できるパターン数にユトリが出，その分作れるゲームにも幅が出てくるということを意味しています．そのため，レジスタの利用法やアドレスの計算方法をまったく変えて，速度だけを追求したプログラムを**リスト2-3**に示します．そして，これから我々が使っていくのも，当然こちらの高速表示のほうです．

このプログラムは，まず，VRAMのアドレス計算方法に工夫を凝らしています．ゲーム座標でのY軸の+1はVRAM上では+280H(10進数では+640バイト)の増加になっていますが，この増分を100Hの倍数の和に分解すると下の式のようになります．

求めるアドレス
=280H×Y
=(200H+80H)×Y
=200H×Y+80H×Y
=2×100H×Y+100H×Y÷2

式そのものは複雑になったように見えますが，プログラム上は，速度のアップにつながるという利点が生まれるのです．

AX，BX，CX，DXの各レジスタはそれぞれ，上位と下位の2つに分けて使うことができるというのはもう知っていますね．この性質をうまく使うことによって，簡単に100H倍の数値を作ることができるのです．もう気づいた方もいると思いますが，100H倍しようとする数値を単純に上位のレジスタに入れるだけで，100H倍したとみなせるのです．

```
MOV  AX, 0  ;AXを0で初期化
MOV  AH, 1  ;AXの上位に1を代入
AXとして全体で考えると100Hが格納されていることになる
```

さらに，さきほどの計算式を実現するには，シフトという考え方を導入することになります．

マシン語の命令のなかには，数値を1桁右，または左にシフトする命令があります．これはコンピュータでは最も基本的な演算の1つですが，しばしば2倍や2で割る演算の代用としても用いられます．

コンピュータが扱っている数値は2進数ですから，この数値を1桁シフトするということは，2進数の桁をずらすことにほかならないのです．上の位のほう，すなわち左に1桁ずらせば(シフトすれば)2倍することになり，右にシフトすれば2で割ることになるのです．もちろん，かけ算や割り算の命令よりもずっと高速に演算されます．

これらを組み合わせて，前ページの式をマシン語で組んだプログラムは次のようになります．

```
MOV  AX, 0      ;AXに0を入れて初期化
MOV  AH, CH     ;Y座標の100H倍の数値を作り出す
SHR  AX, 1      ;AX←AX÷2……Y座標の80H倍に等しい
SHL  CH, 1      ;CH←CH×2……Y座標の200H倍に相当する
ADD  AX, CX     ;AX←280H×Y+X が最終的に求められる
```

ここで，CXの上位のCHにはY座標を，下位のCLにはX座標をあらかじめ格納してあるものとします．

もとのプログラムと比べると，何をしているのか非常にわかりにくいかもしれませんが，今まで使っていたMULという命令と比較すると100クロック以上の節約ですから，実行スピードはかなりアップしたことになるのです．

また，このリスト2-3のプログラムでは，パターン・データ用に専用のセグメント域を設けてありますが，これは必要に応じて自由に設定してください．ここでは，とりあえず8000_Hバイトの大きさにしてあります．

PDADRの部分では，今まで紹介していないBP(Base Pointer)を使っています．BPは主にスタックエリアを使って，サブルーチン間での引数のやりとりに使われるレジスタの一種です．そのため，このBPだけはスタックエリアからデータを取り出しやすいように，セグメント・ベースがSSになっています．したがって，BPを使ってメモリを参照するときには，目的のデータがどこのセグメントに属しているかという注意が必要です．

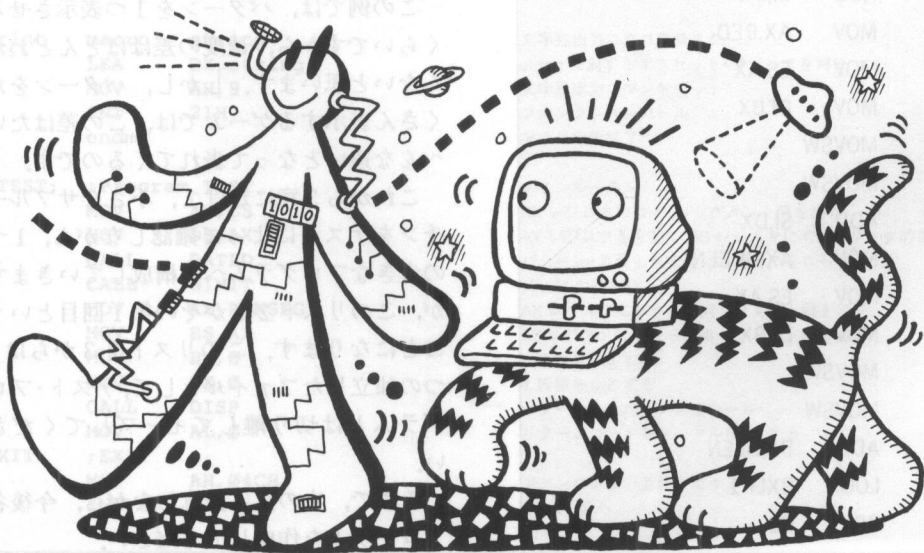
もちろんBPを単なるレジスタとして使うことも可能です．本プログラムでも，BPをデータ・アドレスとして使用しています．

なお，リスト2-3から，**マクロ機能**を使っていきます．プログラムはこのマクロ機能を使うことによって簡潔になり，また，高級言語を使っているかのような感覚にも浸れるのです．

MASMに対する疑似命令やマクロ定義名をCPUに対する命令と区別するために，本書では大文字と小文字とに使い分けをしていますが，とくにそのような決まりがあるわけではありません．

マクロ定義はこの例のようにmacro～endmで囲むことによって定義できます．アセンブラはマクロを定義している名前に出会うと，その部分をあらかじめ定義してあるテキストのブロックに置き換えて，マシン語コードに変換してくれます．

vwrite4というマクロ定義では，マクロ定義内で条件アセンブルを使っています．このように，条件アセンブル(if～endif)と組み合わせることによって，マクロ定義も自由度がかなり上がります．ここではマクロ定義について詳しく説明することはしませんが，じっくりと，これらのプログラムを読んで流れを理解するようにしてください．

では，高速化という観点から，もう一度リスト2-3を見てみましょう．このなかでDISPというサブルーチンがあります．これはパターン表示用のサブルーチンです．このままでは，説明がしにくいので，DISPルーチン内のマクロを具体的な命令に展開したものを次のページに示します．

```
DISP: ; DISPlay
      PUSH   SI
      CALL   PDADR
      CALL   XYADR
      MOV    BX,DISAD
      PUSH   DS
      MOV    AX,PTNSEG
      MOV    DS,AX
      MOV    CX,32
      ADD    BP,4*32
      MOV    DX,NEXTDT-4
BXLP1: ; Box Loop 1
      MOV    SI,BP
      MOV    AX,BLUE
      MOV    ES,AX
      MOV    DI,BX
      MOVSW
      MOVSW
      MOV    BP,SI
      ADD    SI,DX
      MOV    AX,RED
      MOV    ES,AX
      MOV    DI,BX
      MOVSW
      MOVSW
      ADD    SI,DX
      MOV    AX,GREEN
      MOV    ES,AX
      MOV    DI,BX
      MOVSW
      MOVSW
      ADD    BX,HLEN
      LOOP   BXLP1
      POP    DS
      POP    SI
      RET
```

展開してみると，冗長なプログラムになりました．リスト2-2のなかでのDISPルーチンにおける処理と比べてみると，CALL命令を色別に呼んでいるのではなく1つにまとめて処理しています．

さらに高速化をはかるために，ストリング命令のMOVSWを単独で2つ並べて使っています．このとき，REP命令と分離して使っているために，CXにはなんの影響も与えないことに注意してください．

また，このストリング命令で使われるDI, SIは，ディレクション・フラグによって増加，あるいは減少が決定されるのですが，このフラグはDISPルーチンにはいる前にあらかじめCLD命令で+方向と決めてありますので，ここであらためてフラグを操作することはしていません．

この例では，パターンを1つ表示させるくらいですから，速度の差はほとんどわからないと思います．しかし，パターンをたくさん表示するゲームでは，この差はたいへんな違いとなって表れてくるのです．

これから3章にかけて，小さなサブルーチンをテストによって確認しながら，1つの大きなプログラムへと構成していきますが，このリスト2-3がその第1回目ということになります．このリスト2-3からは1つの独立したファイルとして，テスト・プログラムとは切り離してセーブしてください．

そこで，次の点に注意しながら，今後各プログラムを作成してください．

(1) リスト2-3以降のプログラムは，テスト・プログラムから随時，別のファイルとして順番にインクルードしていく構造となる．
(2) インクルードするファイルは独立したファイルとする．
(3) テスト・プログラムの実行に際しては，プログラムのほかにパターン・データが必要になる場合がある．

リスト2-2　パターンの表示(LIST2-2.ASM)

```
;****** List 2-2 *******
VTOP    equ     Ø                    VRAMのトップアドレス
HLEN    equ     8Ø                   1ライン分のVRAMアドレスの増分
HLEN4   equ     8*8Ø                 8ライン分のVRAMアドレスの増分
BLUE    equ     ØA8ØØH               VRAMの青のセグメント値
RED     equ     ØBØØØH               VRAMの赤のセグメント値
GREEN   equ     ØB8ØØH               VRAMの緑のセグメント値
PTNOFF  equ     4*32                 パターンのデータ補正値

CODE    segment                      命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,ES:PTNSEG,SS:STSEG

print   macro   string               文字列出力マクロ定義スタート
        LEA     DX,string            stringに対するオフセット・アドレスを得る
        MOV     AH,9                 文字列出力コマンドセット
        INT     21H                  ファンクションコール
        endm                         マクロ定義終了

PTEST:  ;Program TEST                メインルーチン
        MOV     AX,CS                AXレジスタにコード・セグメント値を格納
        MOV     DS,AX                AXレジスタを介してDSレジスタにCSレジスタの値を代入
        CALL    DATLD                パターン・エディタで作ったデータをロード
        CALL    GINIT                VRAMの初期化
        MOV     AX,PTNSEG            AXレジスタにパターン用セグメント値を格納
        MOV     ES,AX                ESレジスタにパターン用セグメント値を格納
        MOV     BX,Ø                 X座標を0とする
        MOV     CX,Ø                 Y座標を0とする
        CALL    DISP                 パターン表示ルーチンをコール
        MOV     AL,Ø                 リターンコードをノーマル(0)とする
EXIT:   ;EXIT
        MOV     AH,Ø4CH              ファンクション番号をセット
        INT     21H                  MS-DOSへ
```

```
GINIT:  ;Graphic system INITialize
        MOV     AX,4000H            グラフィック画面の表示開始コマンド番号をセット
        INT     18H                 BIOS コール
        MOV     AX,4200H            グラフィック画面モード設定コマンド番号をセット
        MOV     CX,0C000H           640×400 ドット・カラーモードで
        INT     18H                 ROM 内ルーチン・コール
        RET                         リターン

DISP:   ;DISPlay                    ―― B, R, G 各面にパターンを表示
        CALL    XYADR               表示アドレスを求めるため
        MOV     SI,PTNOFF           SI ←パターンデータ補正値
        MOV     AX,BLUE             AX ←ブルー面のセグメント値セット
        CALL    BOX                 データに従って四角形を描く
        MOV     AX,RED              AX ←レッド面のセグメント値セット
        CALL    BOX                 データに従って四角形を描く
        MOV     AX,GREEN            AX ←グリーン面のセグメント値セット
        CALL    BOX                 データに従って四角形を描く
        RET                         リターン

BOX:    ;BOX                        ―― パターンを表示
        MOV     DI,DISPAD           DI ←表示アドレス
        PUSH    DS                  データ・セグメント値退避
        MOV     DS,AX               データセグメント← AX
        MOV     DX,20H              Y 方向ループ回数セット
LOOP1:  ;LOOP 1
        MOV     CX,2                X 方向ループ回数セット
LOOP2:  ;LOOP 2
        MOV     AX,ES:[SI]          AX ←パターンデータ
        MOV     [DI],AX             VRAM へパターンデータをセット
        ADD     DI,2                DI ← DI+2
        ADD     SI,2                SI ← SI+2
        LOOP    LOOP2               2 回ループ
        ADD     DI,HLEN-4           次ラインのアドレスを求める
        DEC     DX                  DX ← DX-1
        JNE     LOOP1               DX=0 でなければ LOOP1 へジャンプ
        POP     DS                  データ・セグメントを復元
        RET                         リターン

XYADR:  ;XY to ADdRess              ―― (BX, CX) から表示アドレスを求める
        PUSH    DX                  DX の値を退避
        MOV     AX,HLEN4            被乗数をセット
        MUL     CX                  DX : AX ← AX×CX
        ADD     BX,AX               BX ← BX+AX
        MOV     DISPAD,BX           表示用アドレスを保存
        POP     DX                  DX レジスタ値を復元
        RET                         リターン
;
DISPAD  dw      0                   DISPlay ADdress
;
DATLD:  ;DATa LoaD
        MOV     SI,offset LODTBL    SI ←ロード・テーブル・アドレス
```

```
DLDLP1: ;Data LoaD LooP 1
        MOV     AL,[SI].CMD              AL←ロード・コマンド
        AND     AL,AL                    コマンドエンドか？
        JE      DLD2                     コマンドエンドであればDLD2へ
        CALL    LOAD                     データ・ロード
        ADD     SI,type lodinf           SIレジスタ更新
        JMP     DLDLP1                   DLDLP1へ
DLD2:   ;Data LoaD 2
        RET
;
lodinf  struc                            ロード情報構造体定義
CMD     db      Ø                        ロード・コマンド
PASS    db      "           "            パス格納領域（11文字分）
ENDSIN  db      Ø                        パス・エンド・コード
LDADR   dw      Ø                        ロード・アドレス
LDSEG   dw      Ø                        ロード・セグメント
lodinf  ends                             構造体定義終了
;
LODTBL  lodinf <1,"PTNDAT1.DAT" ,Ø,Ø,PTNSEG>
        lodinf < >
;
LOAD:   ;LOAD
        LEA     DX,[SI].PASS             ファイルパス名格納アドレス取得
        MOV     AL,Ø                     ファイル・アクセス・コントロール
        MOV     AH,3DH                   ハンドルのオープン
        INT     21H                      ファンクションコール
        JNB     DOPEN                    エラーがなければDOPENへ
        CMP     AX,2                     ファイルが存在しない場合のチェック
        JNE     DLERR                    エラー・コードによる処理の選択
        LEA     DX,[SI].PASS             ファイルパス名格納アドレス取得
        MOV     AH,9                     ストリングのスクリーン出力
        MOV     [SI].ENDSIN,"$"          ストリング終了コードセット
        INT     21H                      ファンクションコール
        print   EMES2                    エラー・メッセージ出力
        MOV     AX,ØFFFFH                エラー・サイン・セット
        JMP     DLRET                    リターン
DLERR:  ;Data Load ERRor
        LEA     DX,[SI].PASS             ファイルパス名格納アドレス取得
        MOV     AH,9                     ストリングのスクリーン出力
        MOV     [SI].ENDSIN,"$"          ストリング終了コードセット
        INT     21H                      ファンクションコール
        print   EMES1                    エラー・メッセージ出力
        MOV     AX,ØFFFFH                エラー・サイン・セット
        JMP     DLRET                    リターン
DOPEN:  ;Data file OPEN
        MOV     HANDL,AX                 ハンドル保存
        MOV     BX,AX                    ハンドルを指定レジスタへ
        MOV     CX,Ø                     CX←0
        MOV     DX,CX                    DX←0
        MOV     AH,42H                   ファイル・ポインタの移動とする
        MOV     AL,2                     ポインタをファイルの終わりに移動とする
```

```
        INT     21H                 ファイルの大きさをAXレジスタへ
        MOV     FLLEN,AX            ファイルの大きさを保存
        MOV     BX,HANDL            ハンドルを指定レジスタへ
        MOV     AH,3EH              ハンドルのクローズ
        INT     21H                 ファンクションコール
        LEA     DX,[SI].PASS        指定ファイルのパス名の格納アドレス取得
        MOV     AL,Ø                ファイル・アクセス・コントロール
        MOV     AH,3DH              ハンドルのオープン
        INT     21H                 ファンクションコール
        MOV     HANDL,AX            ハンドルの保存
        MOV     BX,AX               指定レジスタにハンドルセット
        PUSH    DS                  データ・セグメント値をスタックへ退避
        MOV     DX,[SI].LDADR       ロード・アドレス・セット
        MOV     CX,FLLEN            ファイルの大きさをセット
        MOV     AX,[SI].LDSEG       ロード・セグメント値セット
        MOV     DS,AX               DSへAXを介してセグメント値セット
        MOV     AH,3FH              データ・ロード
        INT     21H                 ファンクションコール
        POP     DS                  DSをスタックから復元
        MOV     FILEL,AX            読み込んだバイト数保存
        MOV     BX,HANDL            ハンドルを指定レジスタへ
        MOV     AH,3EH              ハンドルのクローズ
        INT     21H                 ファンクションコール
        XOR     AX,AX               ノーマル・リターンとする
DLRET:  ;Data Load RET
        RET                         リターン
;
FILEL   dw      Ø                   ┐
FLLEN   dw      Ø                   │ファイル・アクセス用ワークエリア
DISAD   dw      Ø                   │
HANDL   dw      Ø                   ┘
EMES1   db      1Ø,13               エラー・メッセージ番号1
        db      "ファイル オープン エラー"
        db      1Ø,13,"$"
EMES2   db      1Ø,13               エラー・メッセージ番号2
        db      "ファイル ガ, アリマセン "
        db      1Ø,13,"$"
CLEAR   db      1BH,"[2J",1BH,"[>1h",1BH,"[>5h$"
CSRON   db      1BH,"[>5l",1BH,"[>1l$"

CODE    ends                        命令の置かれているセグメントの終わり

STSEG   segment stack               スタック用セグメントの開始
        db      1ØØH dup (?)        データ域を100Hバイト確保
STSEG   ends                        スタック用セグメントの終わり

PTNSEG  segment                     パターン用セグメントの開始
        db     8ØØØH dup (ØFFH)     データ域を8000Hバイト確保
PTNSEG  ends                        パターン用セグメントの終わり

        end                         プログラム・エンド
```

リスト2-3 パターンの表示(高速版)(LIST2-3.ASM)

```
;
;******* List 2-3 *******
;
VTOP     equ     Ø            VRAMのトップアドレス
HLEN     equ     8Ø           1ライン分のVRAMアドレスの増分
NEXTDT   equ     8ØH          NEXT VRAM用増分
NEXTPT   equ     2ØØH         NEXT パターン用増分
BLUE     equ     ØA8ØØH       VRAMの青のセグメント値
RED      equ     ØBØØØH       VRAMの赤のセグメント値
GREEN    equ     ØB8ØØH       VRAMの緑のセグメント値

vwrite4 macro    vramseg                  マクロ定義スタート
        ifidn <vramseg>,<BLUE>            マクロパラメータがBLUEに等しいかチェック
          MOV     SI,BP                   等しければSI←BP
        else
          ADD     SI,DX                   等しくなければSI←SI+DX
        endif                             条件アセンブル・エンド
        MOV     AX,vramseg                AXにマクロパラメータを代入
        MOV     ES,AX                     AXを介してESにセグメント値セット
        MOV     DI,BX                     DIレジスタを初期化
        MOVSW                             ES:[DI]←DS:[SI] & DI←DI+2,SI←SI+2
        MOVSW                             ES:[DI]←DS:[SI] & DI←DI+2,SI←SI+2
        ifidn <vramseg>,<BLUE>            マクロパラメータがBLUEに等しいかチェック
          MOV     BP,SI                   等しければBP←SI
        endif                             条件アセンブル・エンド
        endm                              マクロ定義終了

DISP:   ;DISPlay
        PUSH    SI                        SIレジスタ値をスタックへ退避
        CALL    PDADR                     パターン番号から,データ・アドレスを求める
        CALL    XYADR                     表示アドレスを求めるため
        MOV     BX,DISAD                  表示アドレスをセット
        PUSH    DS                        データセグメント値をスタックへ退避
        assume  DS:PTNSEG
        MOV     AX,PTNSEG                 パターン・データ用セグメント値をセット
        MOV     DS,AX                     データセグメントをセット
        MOV     CX,32                     Y方向ループ回数セット
        ADD     BP,4*32                   透明データ・カットとする
        MOV     DX,NEXTDT-4               次のVRAMへの加算値
BXLP1:  ;BoX LooP 1
        vwrite4 BLUE                      パラメータをBLUEでマクロ展開
        vwrite4 RED                       パラメータをREDでマクロ展開
        vwrite4 GREEN                     パラメータをGREENでマクロ展開
        ADD     BX,HLEN                   次ラインの表示アドレス
        LOOP    BXLP1                     ループをCX回繰り返す
        assume  DS:CODE
        POP     DS                        データ・セグメント復元
        POP     SI                        SIレジスタ復元
        RET                               リターン
```

```
XYADR:   ;XY  to  ADdress                    ── (CL,CH)から表示アドレスを求める
         PUSH    AX                          AXレジスタ値をスタックへ退避
         MOV     AX,Ø                        AXレジスタ初期化
         MOV     AH,CH                       Y座標の100H倍を求める
         SHR     AX,1                        Y座標の80H倍を求める
         SHL     CH,1                        Y座標の200H倍を求める
         ADD     AX,CX                       AX←Y×280H+X
         MOV     DISAD,AX                    表示アドレス保存
         POP     AX                          AXレジスタ復元
         RET                                 リターン

PDADR:   ;Pattern Data ADdRess               ── データ・アドレスをBPレジスタへ求める
         MOV     AH,Ø                        AXの上位アドレスを初期化
         XCHG    AL,AH                       AL←→AH
         SHL     AX,1                        AX←AX×2
         MOV     BP,AX                       パターン・データ・アドレスを求める
         RET                                 リターン

GINIT:   ;Graphic system INITialize
         MOV     AX,4ØØØH                    グラフィック画面の表示開始コマンドをセットする
         INT     18H                         ROM内ルーチン・コール
         MOV     AX,42ØØH                    グラフィック画面モード設定コマンドをセットする
         MOV     CH,ØCØH                     640×400ドット・カラー・モードで
         INT     18H                         ROM内ルーチン・コール
         RET                                 リターン
;
DATLD:   ;DATa LoaD
         MOV     SI,offset LODTBL            SI←ロード・テーブル・アドレス
DLDLP1:  ;Data LoaD LooP 1
         MOV     AL,[SI].CMD                 AL←ロードコマンド
         AND     AL,AL                       コマンドエンドか？
         JE      DLD2                        コマンドエンドであればDLD2へ
         CALL    LOAD                        データ・ロード
         ADD     SI,type lodinf              SIレジスタ更新
         JMP     DLDLP1                      DLDLP1へ
DLD2:    ;Data LoaD 2
         RET
;
LOAD:    ;LOAD
         LEA     DX,[SI].PASS                ファイルパス名格納アドレス取得
         MOV     AL,Ø                        ファイル・アクセス・コントロール
         MOV     AH,3DH                      ハンドルのオープン
         INT     21H                         ファンクションコール
         JNB     DOPEN                       エラーがなければDOPENへ
         CMP     AX,2                        ファイルが存在しない場合のチェック
         JNE     DLERR                       エラー・コードによる処理の選択
         LEA     DX,[SI].PASS                ファイルパス名格納アドレス取得
         MOV     AH,9                        ストリングのスクリーン出力
         MOV     [SI].ENDSIN,"$"             ストリング終了コードセット
         INT     21H                         ファンクションコール
         print   EMES2                       エラー・メッセージ出力
```

```
        MOV     AX,ØFFFFH              エラー・サイン・セット
        JMP     DLRET                  リターン
DLERR:  ;Data Load ERRor
        LEA     DX,[SI].PASS           ファイルパス名格納アドレス取得
        MOV     AH,9                   ストリングのスクリーン出力
        MOV     [SI].ENDSIN,"$"        ストリング終了コードセット
        INT     21H                    ファンクションコール
        print   EMES1                  エラー・メッセージ出力
        MOV     AX,ØFFFFH              エラー・サイン・セット
        JMP     DLRET                  リターン
DOPEN:  ;Data file OPEN
        MOV     HANDL,AX               ハンドル保存
        MOV     BX,AX                  ハンドルを指定レジスタへ
        MOV     CX,Ø                   CX←0
        MOV     DX,CX                  DX←0
        MOV     AH,42H                 ファイル・ポインタの移動とする
        MOV     AL,2                   ポインタをファイルの終わりに移動とする
        INT     21H                    ファイルの大きさをAXレジスタへ
        MOV     FLLEN,AX               ファイルの大きさを保存
        MOV     BX,HANDL               ハンドルを指定レジスタへ
        MOV     AH,3EH                 ハンドルのクローズ
        INT     21H                    ファンクションコール
        LEA     DX,[SI].PASS           指定ファイルのパス名の格納アドレス取得
        MOV     AL,Ø                   ファイル・アクセス・コントロール
        MOV     AH,3DH                 ハンドルのオープン
        INT     21H                    ファンクションコール
        MOV     HANDL,AX               ハンドルの保存
        MOV     BX,AX                  指定レジスタにハンドルセット
        PUSH    DS                     データ・セグメント値をスタックへ退避
        MOV     DX,[SI].LDADR          ロード・アドレス・セット
        MOV     CX,FLLEN               ファイルの大きさをセット
        MOV     AX,[SI].LDSEG          ロード・セグメント値セット
        MOV     DS,AX                  DSへAXを介してセグメント値セット
        MOV     AH,3FH                 データ・ロード
        INT     21H                    ファンクションコール
        POP     DS                     DSをスタックから復元
        MOV     FILEL,AX               読み込んだバイト数保存
        MOV     BX,HANDL               ハンドルを指定レジスタへ
        MOV     AH,3EH                 ハンドルのクローズ
        INT     21H                    ファンクションコール
        XOR     AX,AX                  ノーマル・リターンとする
DLRET:  ;Data Load RET
        RET                            リターン
;
FILEL   dw      Ø                      ┐
FLLEN   dw      Ø                      │ ファイル・アクセス用ワークエリア
DISAD   dw      Ø                      │
HANDL   dw      Ø                      ┘
EMES1   db      1Ø,13                  エラー・メッセージ番号1
        db      "ファイル オープン エラー"
        db      1Ø,13,"$"
```

```
EMES2   db      1Ø,13                                     エラー・メッセージ番号2
        db      "ファイル ガ, アリマセン "
        db      1Ø,13,"$"
CLEAR   db      1BH,"[2J",1BH,"[>1h",1BH,"[>5h$"
CSRON   db      1BH,"[>51",1BH,"[>11$"

CODE    ends                                              命令の置かれているセグメントの終わり

STSEG   segment stack                                     スタック用セグメントの開始
        db      1ØØH dup (?)                              データ域を100Hバイト確保
STSEG   ends                                              スタック用セグメントの終わり

PTNSEG  segment                                           パターン用セグメントの開始
        db      8ØØØH dup (ØFFH)                          データ域を8000Hバイト確保
PTNSEG  ends                                              パターン用セグメントの終わり

        end                                               プログラム・エンド
```

テスト2-3　テスト・プログラム(TEST2-3.ASM)

```
;******* TEST List 2-3 *******

CODE    segment                                      命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,SS:STSEG

print   macro   string                               文字列出力マクロ定義スタート
        LEA     DX,string                            stringに対するオフセット・アドレスを得る
        MOV     AH,9                                 文字列出力コマンドセット
        INT     21H                                  ファンクションコール
        endm                                         マクロ定義終了

PTEST:  ;Program TEST
        CLD                                          ストリング命令用フラグを+方向とする
        MOV     AX,CS                                コード・セグメント値をAXレジスタへ
        MOV     DS,AX                                DSへAXレジスタを介してコード・セグメント値をセット
        CALL    DATLD                                パターン・データ・ロード
        CALL    GINIT                                グラフィック・システムの初期化
        MOV     CX,ØH                                CXに座標(0,0)を代入 …… CH=Y,CL=X
        MOV     AL,Ø                                 パタンン番号0番セット
        CALL    DISP                                 パターン番号0番を座標(0,0)に表示する
        MOV     AL,Ø                                 システム・リターン・コード=0とする
        MOV     AH,Ø4CH                              プロセスの終了とする
        INT     21H                                  ファンクションコール

lodinf  struc                                        ロード情報構造体定義
CMD     db      Ø                                    ロード・コマンド
PASS    db      "           "                        パス格納領域(11文字分)
ENDSIN  db      Ø                                    パス・エンド・コード
LDADR   dw      Ø                                    ロード・アドレス
LDSEG   dw      Ø                                    ロード・セグメント
lodinf  ends                                         構造体定義終了
;
LODTBL  lodinf <1,"PTNDAT1.DAT" ,Ø,Ø,PTNSEG>
        lodinf < >
;
        include LIST2-3.ASM                          LIST2-3.ASMを取り込む
```

5. パターン消去…キャラクタを動かす前に

画面へのパターンの表示が自由にできるようになれば，次はそれを動かしたいと思うのが人間の心理というものです．心理学的にもそれが当然なのではないかと思いますが？

もうだいぶ前のことですが，学生時代にキタナイ格好をしてヨーロッパを放浪していたことがあります．そのとき，ベルギーのブリュッセルにある有名な小便小僧の像を見ようと思い，地図を片手にその近くまでいったのですが，どうしても見つけられませんでした．……実は，余りに小さかったため，何度もその前を往復していたのです．……それで，通りすがりの町の人に聞いたのですが，小便小僧という言葉がわからなかったため，恥ずかしながら道のまんなかで大胆にも小便小僧のマネをしたのです．ジェスチャーは世界共通の言葉です．彼は，「アー，ワカッタ，ワカッタ!!」というような顔をして，それなら道の反対側にあるというのです．「おかしいナ，地図が間違っていたのかナァ……」と思いながらいってみると，なんとそこは公衆便所だったのです．

地元の人にとっては，あんなもの取るに足らないものなのでしょう．そこに，彼が旅行者の心理が読めず，こちらは逆に彼の心理が読めなかった原因があったのかもしれません．しかし，本書はお互いにマシン語ゲーム製作を目指しているわけですから，そのようなギャップなどあるはずがありませんね．期待どおりに(？)パターンの移動へ進んでいきます．

ここでは，パターンを動かすための準備として，画面上でものが動くということはプログラム的にはどういう処理をすればいいか，その原理と次節で使うプログラムを作り，テストをしてみることにします．といっても，我々が管理をしているのはゲーム座標という横80コマ，縦50コマの小さな世界ですし，移動の単位もこの1コマを基準とすればいいので，それほど難しいことではありません．

パターンの移動に際しては，どこに移動するのか，まず方向を数値で決めなければなりません．そこで，ゲーム座標上で移動可能な方向すべてに，図2-7のような方向番号を付けることにします．

これで，たとえばゲーム座標で(10,10)の位置にあるパターンを，方向1に1コマ動かす，というような表現ができるようにな

図 2-7　方向番号

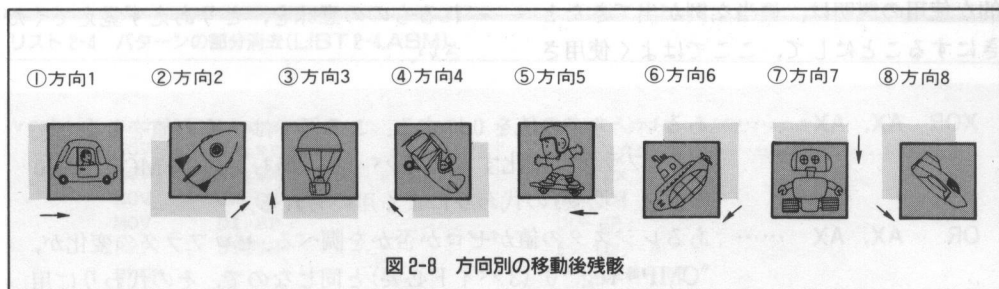



図 2-8　方向別の移動後残骸

りました．この例では，移動後の座標は(11,10)になりますが，では(11,10)の位置に新たにパターンを表示するだけでいいかというと，そうはいきませんね．

前にあったパターンの残骸が，左側に少し残ってしまいます．本書ではパターンのサイズを32×32ドットにしていますから，左，右上，下へ移動する際の，1コマ移動後の残骸を見てみると図2-8のようになります．

結局，移動に際して邪魔になっているのはこの影の部分ですから，移動する前にこの部分だけを消してしまえばいいということです．そのためには，方向別の消去ルーチンを作らなければなりません．そこで，現在位置と移動方向を指示すれば，不要な部分を消去し，次座標を計算した上でその座標がゲームの画面からハズレるか否かも判定してくれるプログラムにすると，便利なものになります．少し長いかもしれませんが，リスト2-4をひととおり読んでください．

リスト2-4の消去ルーチン(CLPTXY)は，リスト2-1の豆腐作りルーチンでVRAMに入れていたFF$_H$を0に変えただけのことです．ただ，サイズが固定では不便なので，消去サイズをCH＝横，CL＝縦として指定できるようにしてあります．

なお，今回のゲームでは，画面のサイズをゲーム座標で(0,0)から(49,49)までとすることにしたので，次に移動する座標がその範囲を越える場合には，ゼロフラグを立ててリターンするようになっています．パターンを表示するときの座標は，パターン左上の座標で示されますから，右端と下端はパターン・サイズを考慮に入れなければなりません．そのため，右端と下端の値は最初からパターン・サイズの分だけ少なくしてあります．また，方向別の不要部分消去後に，移動後の座標が画面から出ないように，それぞれはみ出した値との比較を行っています．したがって，ここでゼロフラグが立てば，その座標は画面外であるということになります．長いプログラムといっても，内容的には同じような処理が8方向分あるだけですから，それほど難しくはないと思います．

しかし，ここで初めて論理演算XORが出てきたので説明を加えておきましょう．

論理演算については，本当に論理演算をするのが目的で使われる場合と，別の目的のために使われる場合とがあります．本格

的な使用の説明は，適当な例が出てきたときにすることにして，ここではよく使用されるものの意味を，とりあえず覚えてください．

XOR　AX，AX　……	あるレジスタの値を0にする．この例ではAXの値を0にしている．フラグは変化するが，2バイトで済むために，MOV reg,0(3バイト必要)の代わりによく用いられる．
OR　AX，AX　……	あるレジスタの値がゼロか否かを調べる．ゼロフラグの変化が，"CMP　reg，0"(3バイト必要)と同じなので，その代わりに用いられる．この例ではAXの値をチェックしている．また，キャリーフラグをリセットしたいときにも使われる．"AND　reg，reg"も同じ意味で使われる．

さて，ここでテスト・プログラムを実行して方向別の消去がうまくされているかどうかを確認してみましょう．なお，テストプログラムの初めには，VRAMのブルー面にデータを埋めるルーチンが挿入されています．テスト・プログラムを実行し，図2-8と同じように消去されていればOKです．

次に，移動後の座標が正しく計算されているかどうかの確認です．これにはマシン語開発には不可欠のデバッガ「SYMDEB」を利用します．これまでのように簡単なプログラムならば，MS-DOSのコマンドラインから直接動かしても，暴走してコントロールがきかなくなることは少ないでしょうが，これから先はそうはいかなくなるかもしれません．こうしてデバッガで確認しておくことが大切です．

では，さっそくSYMDEBを起動してください．

```
A>SYMDEB
-N TEST2-5.EXE
-L
```

準備はOKです．次のように入力してください

```
-G=0 23
-R CX
```

これで，移動後の座標が正しく計算されているかどうかの確認ができます．CXの値をRコマンドでチェックしてください．CXの値が0001と表示されていればCH=0，CL=1のことですから，次の座標が正しく計算されていることになります．同じやりかたで移動方向の初期値を01から08まで順に変更し，全方向について確認してみてください．画面枠からハズレる場合に，表示不可能な座標とゼロフラグがセットされていることが確認できれば，このプログラムは正常に作動しているということです．

リスト2-4　パターンの部分消去(LIST 2-4.ASM)

```
vrstosb macro   vramseg         VRAMへのブロック転送マクロ定義
        MOV     CX,vramseg      CX←マクロ・パラメータ・セット(セグメント値)
        MOV     ES,CX           ES←CX
        MOV     CX,DX           CX←DX
        MOV     DI,BP           DI←BP
        REP     STOSB           ES:[DI]←AL
        endm                    マクロ定義の終了
;
;****** List 2-4-G ******
;
CLPTXY: ;CLear PaTtern  (X,Y)   ── (CL,CH)よりBXのサイズで消去
        MOV     CLXY,BX         CLXYに消去サイズを入れる
        CALL    XYADR           (CL,CH)から消去アドレスを求める
        XOR     AX,AX           AX←0
ERBOX:  ;ERase BOX              ── 指定されたサイズの消去
        MOV     BP,DISAD        BP←消去アドレス
        MOV     BX,DS:CLXY      BX←サイズ
        XOR     DX,DX           DX←0
        XCHG    DL,BH           DL←BH,BH←0
ERL1:   ;ERase Loop 1
        vrstosb BLUE            セグメント値BLUEでマクロ展開
        vrstosb RED             セグメント値REDでマクロ展開
        vrstosb GREEN           セグメント値GREENでマクロ展開
        ADD     BP,HLEN         次ラインの消去アドレスを求める
        DEC     BX              BX←BX−1
        JNE     ERL1            BX=0でなければERL1へ
        RET                     リターン
;
CLXY    dw      Ø               CLXY
;
;****** List 2-4-N ******
;
GRCLS   dw      D1CLS,D2CLS,D3CLS,D4CLS   移動方向別消去ジャンプ・テーブル
        dw      D5CLS,D6CLS,D7CLS,D8CLS

MVCLS:  ;MoVe CLS               ── 移動方向別消去
        MOV     AH,Ø            AH←0
        DEC     AX              AX←AX−1
        SHL     AX,1            AX←AX×2
        ADD     AX,offset GRCLS AX←AX+ジャンプ・テーブル・スタート・アドレス
        MOV     BP,AX           BP←AX
        MOV     AX,DS:[BP+Ø]    AX←移動方向別ベクタ
        JMP     AX              AXへジャンプする
;
D8CLS:  ;Direction 8 CLS        ── 移動方向=8の不要部分消去
        PUSH    CX              CXレジスタ値をスタックへ退避
        MOV     BX,4Ø8H         BX←消去のサイズ
```

```
        CALL    CLPTXY              (CL,CH)よりBXのサイズで消去する
        POP     CX                  CXレジスタの値をスタックから復元
        INC     CH                  CH←CH+1:Y←Y+1
        PUSH    CX                  CXレジスタ値をスタックへ退避
        MOV     BX,118H             BX←消去のサイズ
        CALL    CLPTXY              (CL,CH)よりBXのサイズで消去する
        POP     CX                  CXレジスタ値をスタックから復元
        INC     CL                  CL←CL+1:X←X+1
        CALL    RCHECK              右端チェック
        JE      $+5                 右端であれば(ZF=1)リターン
        CALL    DCHECK              下端チェック
        RET                         リターン
;
D1CLS:  ;Direction 1 CLS            ── 移動方向=1の不要部分消去
        PUSH    CX                  CXレジスタ値をスタックへ退避
        MOV     BX,12ØH             BX←消去のサイズ
        CALL    CLPTXY              (CL,CH)よりBXのサイズで消去する
        POP     CX                  CXレジスタ値をスタックから復元
        INC     CL                  CL←CL+1:X←X+1
        CALL    RCHECK              右端チェック
        RET                         リターン
;
D2CLS:  ;Direction 2 CLS            ── 移動方向=2の不要部分消去
        PUSH    CX                  CXレジスタ値をスタックへ退避
        MOV     BX,118H             BX←消去のサイズ
        CALL    CLPTXY              (CL,CH)よりBXのサイズで消去する
        POP     CX                  CXレジスタ値をスタックから復元
        PUSH    CX                  CXレジスタ値をスタックへ退避
        ADD     CH,3                CH←CH+3:Y←Y+3
        MOV     BX,4Ø8H             BX←消去のサイズ
        CALL    CLPTXY              (CL,CH)よりBXのサイズで消去する
        POP     CX                  CXレジスタ値をスタックから復元
        DEC     CH                  CH←CH−1:Y←Y−1
        INC     CL                  CL←CL+1:X←X+1
        CALL    RCHECK              右端チェック
        JE      $+5                 右端であればリターン
        CALL    UCHECK              上端チェック
        RET                         リターン
;
D3CLS:  ;Direction 3 CLS            ── 移動方向=3の不要部分消去
        PUSH    CX                  CXレジスタ値をスタックへ退避
        ADD     CH,3                CH←CH+3:Y←Y+3
        MOV     BX,4Ø8H             BX←消去のサイズ
        CALL    CLPTXY              (CL,CH)よりBXのサイズで消去する
        POP     CX                  CXレジスタ値をスタックから復元
        DEC     CH                  CH←CH−1:Y←Y−1
        CALL    UCHECK              上端チェック
        RET                         リターン
;
D4CLS:  ;Direction 4 CLS            ── 移動方向=4の不要部分消去
        PUSH    CX                  CXレジスタ値をスタックへ退避
        ADD     CH,3                CH←CH+3:Y←Y+3
```

```
        MOV     BX,4Ø8H             BX←消去のサイズ
        CALL    CLPTXY              (CL,CH)よりBXのサイズで消去する
        POP     CX                  CXレジスタ値をスタックから復元
        PUSH    CX                  CXレジスタ値をスタックへ退避
        ADD     CL,3                CL←CL+3：X←X+3
        MOV     BX,118H             BX←消去のサイズ
        CALL    CLPTXY              (CL,CH)よりBXのサイズで消去する
        POP     CX                  CXレジスタ値をスタックから復元
        DEC     CH                  CH←CH−1：Y←Y−1
        DEC     CL                  CL←CL−1：X←X−1
        CALL    LCHECK              左端チェック
        JE      $+5                 左端(ZF=0)であればリターン
        CALL    UCHECK              上端チェック
        RET                         リターン
;
D5CLS:  ;Direction 5 CLS            ── 移動方向=5の不要部分消去
        PUSH    CX                  CXレジスタ値をスタックへ退避
        ADD     CL,3                CL←CL+3：X←X+3
        MOV     BX,12ØH             BX←消去のサイズ
        CALL    CLPTXY              (CL,CH)よりBXのサイズで消去する
        POP     CX                  CXレジスタ値をスタックから復元
        DEC     CL                  CL←CL−1：X←X−1
        CALL    LCHECK              左端チェック
        RET                         リターン
;
D6CLS:  ;Direction 6 CLS            ── 移動方向=6の不要部分消去
        PUSH    CX                  CXレジスタ値をスタックへ退避
        MOV     BX,4Ø8H             BX←消去のサイズ
        CALL    CLPTXY              (CL,CH)よりBXのサイズで消去する
        POP     CX                  CXレジスタの値をスタックから復元
        PUSH    CX                  CXレジスタ値をスタックへ退避
        ADD     CL,3                CL←CL+3：X←X+3
        INC     CH                  CH←CH+1：Y←Y+1
        MOV     BX,118H             BX←消去のサイズ
        CALL    CLPTXY              (CL,CH)よりBXのサイズで消去する
        POP     CX                  CXレジスタ値をスタックから復元
        INC     CH                  CH←CH+1：Y←Y+1
        DEC     CL                  CL←CL−1：X←X−1
        CALL    LCHECK              左端チェック
        JE      $+5                 左端であればリターン
        CALL    DCHECK              下端チェック
        RET                         リターン
;
D7CLS:  ;Direction 7 CLS            ── 移動方向=7の不要部分消去
        PUSH    CX                  CXレジスタ値をスタックへ退避
        MOV     BX,4Ø8H             BX←消去のサイズ
        CALL    CLPTXY              (CL,CH)よりBXのサイズで消去する
        POP     CX                  CXレジスタの値をスタックから復元
        INC     CH                  CH←CH+1：Y←Y+1
        CALL    DCHECK              下端チェック
        RET                         リターン
```

```
;
REND     equ      46                    右端値
LEND     equ      Ø                     左端値
UEND     equ      Ø                     上端値
DEND     equ      46                    下端値

check    macro    direction,reg         方向別チェックマクロ定義
         MOV      AL,direction          AL←マクロパラメータ
         CMP      AL,reg                方向別チェック
         RET                            リターン
         endm                           マクロ定義終了

RCHECK:  check    REND+1,CL             右端チェック・マクロ展開
LCHECK:  check    LEND-1,CL             左端チェック・マクロ展開
UCHECK:  check    UEND-1,CH             上端チェック・マクロ展開
DCHECK:  check    DEND+1,CH             下端チェック・マクロ展開

         include  LIST2-3.ASM           LIST2-3.ASMを取り込む
```


テスト2-4 テスト・プログラム(TEST 2-4.ASM)

```
;****** List 2-4-T ******

CODE     segment                        命令の置かれているセグメントの始まり

         assume CS:CODE,DS:CODE,SS:STSEG

print    macro    string                文字列出力マクロ定義スタート
         LEA      DX,string             stringに対するオフセット・アドレスを得る
         MOV      AH,9                  文字列出力コマンドセット
         INT      21H                   ファンクションコール
         endm                           マクロ定義終了
;
PTEST:   ;Program TEST
         CLD                            ディレクション・フラグ・リセット
         MOV      AX,CS                 AX←CS
         MOV      DS,AX                 AXレジスタを介してDSにCSを格納
         CALL     GINIT                 グラフィック・システム初期化
         MOV      AX,BLUE               AX←BLUE
         MOV      ES,AX                 AXレジスタを介してESにBLUEをセット
         MOV      DI,Ø                  DI←0
         MOV      CX,78ØH               ストリング命令用ループ回数セット
         MOV      AX,ØFFFFH             AX←FFFFH
```

```
         REP     STOSW                    ES：[DI]← AX
         MOV     AX,1                     AX ← 1：移動方向初期値
         MOV     CX,ØØØØ                  現在位置(CL, CH)←(0, 0)
TLOOP:   ;Test LOOP
         PUSH    AX                       AX レジスタ値をスタックへ退避
         PUSH    CX                       CX レジスタ値をスタックへ退避
         CALL    MVCLS                    移動方向別の消去を行うため
         POP     CX                       CX レジスタ値をスタックから復元
         POP     AX                       AX レジスタ値をスタックから復元
         ADD     CL,5                     CL ← CL+5：X 座標を+5 する
         INC     AX                       AX ← AX+1
         CMP     AX,9                     移動方向エンドか
         JNE     TLOOP                    移動方向エンドでなければ TLOOP へ
         MOV     AL,Ø                     ノーマル・エンド
         MOV     AH,4CH                   コマンド・セット
         INT     21H                      ファンクションコール(MS-DOS へ戻る)

lodinf   struc                            ロード情報構造体定義
CMD      db      Ø                        ロード・コマンド
PASS     db      "           "            パス格納領域(11 文字分)
ENDSIN   db      Ø                        パス・エンド・コード
LDADR    dw      Ø                        ロード・アドレス
LDSEG    dw      Ø                        ロード・セグメント
lodinf   ends                             構造体定義終了
;
LODTBL   lodinf <1,"PTNDAT1.DAT" ,Ø,Ø,PTNSEG>
         lodinf < >
;

         include LIST2-4.ASM              LIST2-4.ASM のファイルの取り込み
```

6. パターン移動…データにそって移動

パターンを動かすために必要な準備は整いました．さて，どのように動かしたらいいでしょうか．つまり，勝手に動けといっても，コンピュータは命令がなければ何もできないのはご存じのとおりです．日本人の習慣で「適当にたのむ……」という言葉が，寿司屋とか酒場などでよく聞かれますが，これが通用するのは店のほうで最初から『適当に』というメニューを用意しているからですね．よく考えると，こんな恐ろしい言葉はナイのですが，日本人は謙虚な人種ですから，決して一番高い料理を出して大儲けをしようなどとは思わないのです．それよりも，また来てもらったほうがいいということを知っているのです．

コンピュータにも，この『適当に』が通用するようになると，本当に便利なのですが，残念ながら無理なのです．そこで，まずパターンに画面のなかをグルグル回ってもらうことにしました．リスト2-5を見てください．ここでの処理はすべてテスト・プログラム中で行われています．基本的な考え方はBASICでデータ文を読むのと同じことで，スタート地点から次に移動する方向をすべてデータとして用意しているのです．たったこれだけのことですから，このパターンは画面のなかをいつまでもグルグル回り続けることになります．

これでは，終わりがなく暴走しているようなものですから，とりあえず15回転したならばストップするようにカウンタを付けました．

このようなデータのことを「テーブル」ともいい，うまく利用すると計算式では求められない複雑な動きを，簡単な上，高速に実現させることができます．これは，ゲームキャラクタ・パターンに性格付けする場合にたいへん有効なテクニックの1つです．なお，プログラム中のデータ作成は，このように実際の数値だけでなく，ラベルで代用できますから，アセンブラの使い方として覚えておくと便利です．

さて，実際にテストの実行をすると，これまでと違ってカーソルとテキスト画面の邪魔な文字が消えています．このカーソルとテキスト画面を消すという作業は，MS-DOSのファンクションコールのディスプレイ・ストリングで実現しています．また，プログラムの終了部分ではカーソルのオンをしていることに注意してください．

リスト 2-5　データによるパターンの移動(TEST 2-5.ASM)

```
;******* TEST List 2-5 *******
CODE    segment                          命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,SS:STSEG

print   macro   string                   文字列出力マクロ定義スタート
        LEA     DX,string                stringに対するオフセット・アドレスを得る
        MOV     AH,9                     文字列出力コマンドセット
        INT     21H                      ファンクションコール
        endm                             マクロ定義終了
;
PTEST:  ;Program TEST
        CLD                              ディレクション・フラグ・リセット
        MOV     AX,CS                    AX ← CS
        MOV     DS,AX                    AXレジスタを介してDSにCSを格納
        print   CLEAR                    テキスト画面クリア
        CALL    DATLD                    パターン・データ・ロード
        CALL    GINIT                    グラフィック・システム初期化
        MOV     AX,10H                   AX ← 10H
        MOV     COUNT,AX                 カウンター値初期化
        MOV     CX,1419H                 パターンの初期座標
TINIT:  ;Test INITialize
        MOV     BX,offset DATA           BX ←方向データの先頭アドレス
        MOV     DATAWK,BX                方向データの先頭アドレス保存
        MOV     AX,COUNT                 AX ← COUNT
        DEC     AX                       AX ← AX−1
        MOV     COUNT,AX                 COUNT ← AX
        JNE     TLOOP                    COUNT＝0でなければTLOOPへ
        print   CSRON                    カーソル・オン
        MOV     AX,04C00H                コマンド・セット
        INT     21H                      MS-DOSへ
;
TLOOP:  ;Test LOOP
        MOV     BX,DATAWK                BX ← DATAWK …… 方向データ・ポインタ
        MOV     AL,[BX]                  AL ←移動方向を示すデータ
        INC     BX                       BX ← BX+1
        MOV     DATAWK,BX                DATAWK ← BX
        OR      AL,AL                    AL＝0か？
        JE      TINIT                    AL＝0であればTINITへ
        CALL    MVCLS                    移動方向別の消去を行う
        PUSH    CX                       CXレジスタ保存
        MOV     AL,0                     AL ← 0(パターン番号)
        CALL    DISP                     (CL, CH)にパターン番号ALを表示する
        POP     CX                       CXレジスタ値復元
        JMP     TLOOP                    TLOOPへ
```

```
;
COUNT   dw      1                                   COUNTer
DATAWK  dw      Ø,Ø                                 DATA WorK area
;
QRR     equ     1
QUR     equ     2
QUU     equ     3
QUL     equ     4                                   方向のラベル化
QLL     equ     5
QDL     equ     6
QDD     equ     7
QDR     equ     8
;
DATA:   ;direction DATA                             ── 移動方向を示すデータ
        db      QDD,QDD,QDD,QDR
        db      QDD,QDD,QDR,QDD
        db      QDR,QDR,QDR,QRR
        db      QDR,QDR,QRR,QDR
        db      QRR,QRR,QRR,QUR
        db      QRR,QRR,QUR,QRR
        db      QUR,QUR,QUR,QUU
        db      QUR,QUR,QUU,QUR
        db      QUU,QUU,QUU,QUL
        db      QUU,QUU,QUL,QUU
        db      QUL,QUL,QUL,QLL
        db      QUL,QUL,QLL,QUL
        db      QLL,QLL,QLL,QDL
        db      QLL,QLL,QDL,QLL
        db      QDL,QDL,QDL,QDD
        db      QDL,QDL,QDD,QDL
        db      Ø                                   0はデータの終了を意味する
;
lodinf  struc                                       ロード情報構造体定義
CMD     db      Ø                                   ロード・コマンド
PASS    db      "            "                      パス格納領域（11文字分）
ENDSIN  db      Ø                                   パス・エンド・コード
LDADR   dw      Ø                                   ロード・アドレス
LDSEG   dw      Ø                                   ロード・セグメント
lodinf  ends                                        構造体定義終了
;
LODTBL  lodinf <1,"PTNDAT1.DAT" ,Ø,Ø,PTNSEG>
        lodinf < >
;
        include LIST2-4.ASM
```

7. 大量出現…一人じゃつまんない！

1つのパターンが動けば，次は数を増やしたくなるのがこれまた人間の欲というか，心理です．諺にもありましたね……，『這えば立て，立てば歩めの親心』……マァ，それほどの可愛さではないにしても，自分で作成したパターンが自分の思いどおりに動いてくれるということは，ある種の感動があるものです．自分の子供でさえも，これほど思いどおりには動いてくれませんからね．それどころか，段々反抗的にさえなるのですから，画面のなかのこの小さなパターンとは大違いです．もっとも，いつまでたっても命令どおりのことしかできない人間では，教えるほうもめんどうでたまりません．コンピュータも同じことで，そのために自己学習能力のある人工知能を開発しているくらいです．程度こそ違え，少しずつパターンが成長していくということは，非常にうれしいものです．

ここでのプログラムは，これまでのものに比べかなり実際のゲームを意識して作られています．つまり，この段階では不必要なものも，現実のゲーム・プログラムに近づけるために入れてあるのです．そのことを頭に入れた上で，まずは大量出現のための秘密兵器，新しい概念の登場です．それは，構造体という概念です．

といっても，ただのデータの集まりのことですので，難しいことではありません．基本的な構造体の定義方法は次のようになります．

```
構造体名      struc
フィールド名  db  式
              dw
   ：         ：   ：
フィールド名  dd  式
   〃         dt  〃
   〃         dq  〃
構造体名      ends
(変数名)構造体名<(初期化リスト)>
              注：( )内は省略可能
```

敵にしても味方にしても，現在の座標やさまざまな情報(画面に出現しているとか弾に当たったなど)を保存するためのメモリ・ブロックが必要です．これは，一般にワークエリアといいますが，このメモリの基本的な構造を，1つの構造体として定義するのです．

```
敵情報のワークエリア用構造体の定義
tinfo    struc
STATUS   db  0  ；敵の情報
PATTERN  db  0  ；パターン番号
XZAHYOU  db  0  ；X 座標
YZAHYOU  db  0  ；Y 座標
POINTER  dw  0  ；ポインタ
TENSUU   dw  0  ；得点
DUMMY    dw  4 DUP(?)；予備
tinfo    ends
```

このように構造体に定義しておくとメモリの参照がとてもわかりやすくなります．ここでは3つの敵（勝手に動くということはすなわち敵となる）を出現させ，パターンも変えることにしました．実際に使う変数は次のように定義できますから，とてもすっきりとしています．

```
ENEMY1  tinfo  <1, 1, 0BH, 10H, offset DATA, 10>
ENEMY2  tinfo  <1, 2, 1, 1FH, offset DATA, 10>
ENEMY3  tinfo  <1, 3, 15H, 1FH, offset DATA, 10>
```

ここでわかりにくいのは，データの参照方法です．これについては，実際の例を見てもらったほうがいいでしょう．たとえば，ENEMY1（敵）のX座標を参照したい場合には

```
MOV  AL, ENEMY1.XZAHYOU
```

または，インデックスレジスタを使って，

```
LEA  SI, ENEMY1
MOV  AL, [SI].XZAHYOU
```

となります．ここで「.」は**ストラクチャフィールド名演算子**というものです．

さて，これだけあるワークエリアのうち，ここで必要なのはパターン番号，座標，方向データ・ポインタの3つだけなのですが，実際のゲーム（といっても，2，3章だけで作るミニミニゲームの話）で必要になると思われるものも用意されています．

ワークエリアの内容とラベルの意味がわかると，プログラムが理解しやすくなりますから，よく確認しておいてください．

なお，ここでは使いませんが，**リスト2-6**の方向を示すデータに9と10という新たな番号が加わっています．9は移動ナシ，10は方向データ・ポインタを新しく変更することを意味しています．

敵が1コマ移動するまでの全体の流れは，**図2-9**のようになっています．プログラムだけを追いかけると，どうしても視野が狭くなってしまい，全体がボケてしまうことがあります．プログラムも全体の内容を考えながら，1行ごとの命令を組み立てていくようにするといいでしょう．

もう1つの新しいテクニックは，プログラムを［ESC］によって終了させていることです．このキースキャンの方法については，次節で詳しく取り上げていますので，ここは早速テスト・プログラムの実行に移りましょう．3機編隊の敵が，画面のなかをグルグル回りだしています．移動方向データを変えれば，色々な動きをさせることも可能ですから，遊んでみるのもいいかもしれません．しかし，まだ画面からはみ出したときの処理はされていませんので，あまり無理はさせないようにお願いします．

リターン
（コール先へ戻る）

出現していない

敵が出現中か否かをチェック

爆発中

EMDEAD
〔リスト3-5〕

出現中

・パターンの表示
・主人公の弾と敵との衝突チェック
〔リスト3-5〕+リターン

移動なし(AL=9)

・方向データ・ポインタの値ALを調べる
・次回のために，方向データ・ポインタを1つ進めておく

方向データ・ポインタを変更(AL=10)

方向データ・ポインタを次の2バイトのデータとする

画面内

・不要部分の消去
・次座標の計算
・次座標のチェック

画面外

・パターンの消去
・敵出現フラグのリセット+リターン

図2-9　敵が1コマ移動するルーチンの流れ

リスト 2-6　パターンの大量出現(LIST 2-6.ASM)

```
;
;****** List 2-6-N ******
;
EMMOVE: ;EneMy MOVE                          ―― すべての敵の移動
        MOV     AL,[SI].TSTATUS              AL ←[SI].TSTATUS
        OR      AL,AL                        AL の 0 チェック
        JNE     $+3                          } AL が 0 であればリターン
        RET
        ifdef   EMDEAD                       EMDEAD が定義されていれば以下をアセンブル
          INC     AL                         AL ← AL+1
          JNE     NEMDEAD                    0 でなければ NEMDEAD へ
          JMP     EMDEAD                     EMDEAD
        endif                                条件アセンブル終了
NEMDEAD:;Not EneMy DEAD
        MOV     CX,word ptr [SI].XZAHYOU     CX ←(X, Y)
        MOV     BX,[SI].POINTER              BX ←ポインタ
EMØ:    ;Enemy Move-Ø
        MOV     AL,[BX]                      AL ← DS：[BX]
        INC     BX                           BX ← BX+1：ポインタ更新
        MOV     [SI].POINTER,BX              ポインタ保存
        CMP     AL,9                         AL=9 か？
        JE      EM1                          AL=9 であれば EM1 へ
        CMP     AL,1Ø                        AL=10 か？
        JNE     NEM2                         AL=10 でなければ NEM2 へ
        MOV     BX,[BX]                      新しいポインタを取得
        JMP     EMØ                          EM0 へジャンプ
NEM2:   ;Not Enemy Move 2
        CALL    MVCLS                        移動方向別消去
        JNE     NEMFIN                       ZF=1 でなければ NEMFIN へ
        JMP     EMFIN                        EMFIN
NEMFIN: ;Not Enemy Move FINish
        MOV     word ptr [SI].XZAHYOU,CX     座標(X, Y)の保存
EM1:    ;Enemy Move 1
        MOV     AL,[SI].PATTERN              AL ←パターン番号
        CALL    DISP                         DISP をコールし(CL, CH)にパターン AL を表示
        ifdef   EXPLO1                       EXPLO1 が定義されていれば以下をアセンブル
          CALL    EMCHK                      主人公の弾との衝突チェックのため
          JB      $+3                        衝突していなければリターン
          RET                                リターン
          DEC     BX                         BX ← 0
          MOV     byte ptr [BX],Ø            弾の出現フラグを 0 にする
          MOV     [SI].TSTATUS,ØFFH          敵の状態を爆発中とする
          MOV     [SI].PATTERN,EXPLO1        パターン番号を爆発のパターンとする
          MOV     DX,[SI].TOKUTEN            DX ←敵の得点を格納
          CALL    DISPSC                     スコアの加算表示
        endif                                条件アセンブルの終了
        RET                                  リターン
```

```
;
EMFIN:  ;Enemy Move FINish
        MOV     CX,word ptr [SI].XZAHYOU        CX←座標(X,Y)
        MOV     BX,420H                         BX←消去のサイズ
        CALL    CLPTXY                          (CL,CH)よりBXのサイズで消去
        MOV     [SI].TSTATUS,0                  敵の出現フラグを0とする
        RET                                     リターン
;
EMVAL   equ     3                               EneMy VALue
;
EMMVAL: ;EneMy MoVe ALl
        MOV     SI,offset ENEMY                 敵ワークエリア先頭アドレス
        MOV     CX,EMVAL                        CX←敵の総数
EMALP:  ;Enemy Move ALL Loop
        PUSH    CX                              CXの値をスタックへ退避
        CALL    EMMOVE                          敵の移動
        ADD     SI,type ENEMY                   敵1機のワークエリアの長さ
        POP     CX                              CXの値をスタックから復元
        LOOP    EMALP                           CX回ループする
        RET                                     リターン
;
tinfo   struc                                   敵ワークエリア用構造体定義
        TSTATUS db 0                            状態
        PATTERN db 0                            パターン番号
        XZAHYOU db 0                            X座標
        YZAHYOU db 0                            Y座標
        POINTER dw 0                            ポインタ
        TOKUTEN dw 0                            得点
        DUMMY   dw 4 dup (?)                    予備
tinfo   ends                                    構造体定義の終わり

ENEMY   tinfo   <>                              敵ワークエリア3機分確保
        tinfo   <>
        tinfo   <>

        include LIST2-4.ASM                     LIST2-4.ASMファイルを取り込む
```

テスト 2-6　テスト・プログラム(TEST 2-6.ASM)

```
;****** List 2-6-T ******

CODE    segment                         命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,ES:PTNSEG,SS:STSEG

print   macro   string                  文字列出力マクロの定義
        LEA     DX,string               マクロパラメータのオフセットを DX へ
        MOV     AH,9                    ファンクションコール番号9 …… 文字列出力
        INT     21H                     ファンクションコール
        endm                            マクロ定義終了
;
PTEST:  ;Program TEST
        CLD                             ディレクション・フラグ・リセット
        MOV     AX,CS                   AX ← CS
        MOV     DS,AX                   AX レジスタを介して DS に CS を格納
        print   CLEAR                   テキスト画面クリア
        CALL    DATLD                   パターン・データ・ロード
        CALL    GINIT                   グラフィック・システム・クリア
        MOV     SI,offset ENEMY1        敵,初期データ先頭アドレス
        MOV     DI,offset ENEMY         敵,ワークエリア先頭アドレス
        MOV     CX,type ENEMY*3         敵用テーブル×3
        MOV     AX,CS                   AX ← CS
        MOV     ES,AX                   ES ← AX
        REP     MOVSB                   ES : [DI]← DS : [SI]
;
TLOOP:  ;Test LOOP
        CALL    EMMVAL                  すべての敵を移動するため
        MOV     AL,Ø                    キーコード・グループ番号を0とする
        MOV     AH,Ø4H                  コマンド番号4セット
        INT     18H                     キーデータ取り込み
        ROR     AH,1                    右へローテートしてキャリーへキーデータを取り込む
        JNB     TLOOP                   [ESC] が押されていなければ TLOOP へ
        print   CSRON                   カーソル・オン
        MOV     AX,ØCØØH              }
        INT     21H                   } キーボード・バッファ・クリア
        MOV     AL,Ø                    リターン・コード←ノーマル(0)
        MOV     AH,Ø4CH                 エンド・プロセス・コマンド
        INT     21H                     ファンクションコール
;
QRR     equ     1                     }
QUR     equ     2                     }
QUU     equ     3                     }
QUL     equ     4                     }
QLL     equ     5                     }
QDL     equ     6                     } 方向のラベル化
QDD     equ     7                     }
QDR     equ     8                     }
NM      equ     9                     }
NP      equ     1Ø                    }
```

```
;
DATA:   ;direction DATA                          ── 移動方向テーブル
        db      QDD,QDD,QDD,QDR
        db      QDD,QDD,QDR,QDD
        db      QDR,QDR,QDR,QRR
        db      QDR,QDR,QRR,QDR
        db      QRR,QRR,QRR,QUR
        db      QRR,QRR,QUR,QRR
        db      QUR,QUR,QUR,QUU
        db      QUR,QUR,QUU,QUR
        db      QUU,QUU,QUU,QUL
        db      QUU,QUU,QUL,QUU
        db      QUL,QUL,QUL,QLL
        db      QUL,QUL,QLL,QUL
        db      QLL,QLL,QLL,QDL
        db      QLL,QLL,QDL,QLL
        db      QDL,QDL,QDL,QDD
        db      QDL,QDL,QDD,QDL
        db      NP
        dw      offset DATA
;
testinf struc                                    敵初期化用データ構造体定義
        db Ø                                       状態
        db Ø                                       パターン番号
        db Ø                                       X座標
        db Ø                                       Y座標
        dw Ø                                       ポインタ
        dw Ø                                       得点
        dw 4 dup (?)                               予備
testinf ends                                     構造体定義の終わり

ENEMY1  testinf <1,1,11,1ØH,offset DATA,Ø,>      敵番号1の初期データ
ENEMY2  testinf <1,2, 1,1FH,offset DATA,Ø,>      敵番号2の初期データ
ENEMY3  testinf <1,3,2Ø,1FH,offset DATA,Ø,>      敵番号3の初期データ
;
lodinf  struc                                    ロード情報構造体定義
CMD     db      Ø                                ロード・コマンド
PASS    db      "           "                    パス格納領域（11文字分）
ENDSIN  db      Ø                                パス・エンド・コード
LDADR   dw      Ø                                ロード・アドレス
LDSEG   dw      Ø                                ロード・セグメント
lodinf  ends                                     構造体定義終了
;
LODTBL  lodinf <1,"PTNDAT1.DAT" ,Ø,Ø,PTNSEG>
        lodinf < >
;
        include LIST2-6.ASM                      LIST2-6.ASMファイルを取り込む
```

8. キー入力…コントロール&ショット

画面のなかを勝手に動いているパターンは，どうやっても主人公にはできませんから，必然的に敵ということになります．しかし，ときには主人公が勝手に動いて，こちらはその他大勢いる敵，というゲームがあってもいいと思うのですが……．大体，普通のゲームでは，敵はいくらでもいるのに，こちらは多くても5人または5機というふうに，たいへんなハンディを背負っているわけです．これでは，ゆとりを持ってゲームを楽しむことはできません．ここで言う『ゆとり』とは，たとえばテレビの実況生放送では，放送終了時間間際になると，いつ「放送時間がなくなりましたので，たいへん残念ですが……」となるか不安ですが，録画放送ならば安心して楽しみながら見ていられるというようなものです．わかる人にしかわからない，飛躍した例になってしまいましたが，せめて自作のゲームぐらいは死なないようにするとかして，『ゆとり』を持ちたいものですね．

さて，やはり敵ばかりではゲームとして成立しませんので，キー操作によってコントロールできる主人公が必要です．ついでに，敵を倒すための小道具として弾と，やられたとき，あるいは敵を倒したときの爆発パターンも作成しておくことにしましょう．

パターンは巻頭口絵4のとおりです．作成したデータはまとめて1つのデータファイルとしてセーブします．

では，リスト2-7を参照してください．

キー操作によって主人公を動かすにはどのキーが押されたかを判定できればいいわけです．これには色々な方法があります．

最も簡単な方法は，PC-9801内のROM内ルーチンを使う方法です．

```
MOV  AL, □  ；キーコードグループ番号
MOV  AH, 4
INT  18H    ；ROM内ルーチンのコール
```

ここで実行していることは，次のような内容です．

①パラメータとしてALにキーコード・グループ番号を入れる．

②AHにファンクションコール番号の4（指定したキーコード・グループ番号の情報をとってくる）を入れてソフトウェア割り込み番号18Hを実行．

③AHに指定したキーコードグループの押された状態をビット情報として返してくれる．

このAHに返されるデータの内容は，押されているキーのビットが1，押されていないキーのビットが0になるようになっています．ですから，返されたデータをビット・チェックすれば，押されたか否かの判断ができることになります．

このほかにはキー入力に対する割り込みを利用する方法もあります．これについては後述することとして，話を先に進めることにしましょう．

ビット

グループ	7	6	5	4	3	2	1	0
0	, ャ / 7 ヤ	& ォ / 6 オ	% ェ / 5 エ	$ ゥ / 4 ウ	# ァ / 3 ア	" / 2 フ	! / 1 ヌ	ESC
1	TAB	BS	¦ / ¥ ー	‘ / ^ へ	= / ー ホ	ヲ / 0 ワ	) ョ / 9 ヨ	(ュ / 8 ユ
2	I ニ	U ナ	Y ン	T カ	R ス	ィ / E イ	W テ	Q タ
3	D シ	S ト	A チ	↵	{ 「 / [。	~ / @ ゛	P セ	O ラ
4	* / : ケ	+ / ; レ	L リ	K ノ	J マ	H ク	G キ	F ハ
5	M モ	N ミ	B コ	V ヒ	C ソ	X サ	ッ / Z ツ	} 」 /] ム
6	ROLL DOWN	ROLL UP	XFER	SPACE	＿ / ロ	? ・ / ／ メ	> 。 / . ル	< 、 / , ネ
7	HELP	HOME CLR	↓	→	←	↑	DEL	INS
8	5	4	*	9	8	7	／	-
9	,	0	=	3	2	1	+	6
A							NFER	.
B								
C	f･6	f･5	f･4	f･3	f･2	f･1	COPY	STOP
D					f･10	f･9	f･8	f･7
E				CTRL	GRAPH	カナ	CAPS	SHIFT
F								

図 2-10　入力ポート別キーボードマップ

最上位ビットや最下位ビットのチェックにはESCの判定でやったように，ビット・シフトで，その値をキャリー・フラグに入れて判定をします．こうすればTEST命令でチェックするより若干(2クロックぶん)スピードが速いからです．わずかな節約ですが，マシン語の場合よく使われます．また，ここでは弾の連続撃ちができないように，前のキーデータを保存しておいて，押し直しがあったときだけ有効とするなど，実践的なテクニックも入れています．

なお，このプログラムでは弾は一度に2発，トータルで画面上には6発まで出るように設定しています．移動用としては，テンキーの4と6がそれぞれ左方向，右方向に対応しています．

ここでキーの操作性を上げるため，プログラムで次のような条件を満たすにはどうすればいいかを考えてみましょう．

1. 4だけが押されているときには左へ移動する．
2. 6だけが押されているときには右へ移動する．
3. 両方とも，押されたときには最後に押されたキーを優先する．

この1と2は問題ありませんが，3番目の両方とも押されたときの処理がなかなか簡単にはできそうにありません．そこで次のように考え方を整理してみます．

キーの入力値が前回と異なれば，何が変化したのかチェックし，そのキーに対応した移動を行う．

実際にこれをプログラム化するには論理演算に関しての知識が必要となってきます．この論理演算には，次に示すように，OR, AND, XOR, TESTの4種類があります．

1. OR(論理和)：第1,第2のどちらかのオペランドのビットが1であれば演算結果のビットを1にする．両方のビットが0ならばそのビットを0にする．一般に第1オペランドの特定のビットを1にしたいときに用いられることが多い．

例：AL＝6BHとBL＝C2HのORをとる

```
    6BH＝01101011
OR  C2H＝11000010
─────────────────
    11101011＝EBH
```

この演算結果がALに格納される
BLの値はC2Hで変化しない．

2. AND(論理積)：第1,第2オペランドの両方のビットが1ならば，演算結果のビットを1にする．どちらか一方が0であればそのビットを0にする．ORとは逆に，第1オペランドの特定のビットを0にしたいときに用いられることが多い．

例：AL＝D2HとBL＝07HのANDをとる

```
     D2H＝11010010
AND  07H＝00000111
──────────────────
     00000010＝02H
```

この演算結果がALに格納される
BLの値は07Hで変化しない

3. XOR(排他的論理和)：第1,第2オペランドのビットが同じならばそのビットを0に，違っていたらそのビットを1にする．主に，第1オペランドのビットを反転させるときに使われる．XORは2度繰り返すともとの値に戻るという特徴がある．

例：AL＝2FHとBL＝16HのXORをとる

```
     2FH＝00101111
XOR  16H＝00010110
──────────────────
     00111001＝39H
```

この演算結果がALに格納される
BLの値は16Hで変化しない

4. TESTという命令の演算はANDと同じだが演算結果をフラグだけに反映させるもので，主にビット・チェックに対して用いられる．

このほかには，NOT(オペランドの全ビットを反転させる演算)という命令があります．

また，論理演算はこのように実際に演算した結果が欲しい場合と，ビットチェックとしてフラグの変化を見るためにする場合とがあります．もちろん，両方を目的として使用することもできますから，利用次第ではたいへん便利なものといえます．ですから，プログラム中に論理演算が出てきた場合には，まず何のために論理演算をして

いるのか，その使用目的を把握することが，プログラムを理解する上で大切なポイントになるのです．

さて，さきほどの問題に戻りましょう．リスト2-7のKEYルーチンを参照してください．ここでは，前回のキー入力値と，現在のキー入力値に対して，チェックのためにXORを使っています．なぜ，XORを使ったのでしょうか？

XORを使えば，まったく同じキー入力であれば演算結果は0となって，ゼロフラグがセットされます．また，どこか異なるビットがあれば，変化したビットだけが1になります．そして，この演算結果ともとのデータとのANDを施すことによって，演算結果が0になれば，0から1への変化すなわち押されたときの変化であることを表し，0にならなければ，1から0への変化，すなわち離されたときの変化であることがわかるのです．

いったい，何をいいたいのかわからないという批判が聞こえそうですが，これらの処理によって，時間の経過すなわち，2つのキーが同時に押されている場合，最後に押されたキーが何かを判断するという問題が，簡単にプログラム化できることになるのです．

実際にこのプログラムの流れを追ってみるとXORとAND命令がかなり重要な役割をしているのがわかっていただけると思います．

2章では，これらの論理演算とキーボードからの入力方法がわかれば，もう卒業です．テスト・プログラムを実行して，しばらくは楽しんでください．

リスト2-7　キー入力による移動と弾の発射(LIST 2-7.ASM)

```
;
;****** List 2-7-G ******
;
vmovsb  macro   vramseg             VRAM 用ブロック転送マクロ定義
        ifidn   <vramseg>,<BLUE>    vramseg が BLUE に等しければ以下をアセンブル
          MOV     SI,DX             SI ← DX
        else                        条件に合わなければ以下をアセンブル
          ADD     SI,NEXTDT-1       SI ← SI＋NEXTDT−1
        endif                       条件アセンブルの終了
        MOV     AX,vramseg          AX ←マクロ・パラメータ
        MOV     ES,AX               AX を介して ES にセグメント値をセット
        MOV     DI,BX               VRAM 格納アドレス
        MOVSB                       ES：[DI]← DS：[SI], DI ← DI+1, SI ← SI+1
        endm                        マクロ定義終了
;
BLPUT:  ;BuLlet PUT                 ── 弾の表示
        CALL    XYADR               表示アドレスを求める
        MOV     BX,DISAD            BX ←表示アドレス
        PUSH    DX                  DX レジスタ値をスタックへ退避
        PUSH    SI                  SI レジスタ値をスタックへ退避
        PUSH    DS                  DS レジスタ値をスタックへ退避
        MOV     AX,PTNSEG           パターン・データ用セグメント値セット
        MOV     DS,AX               AX レジスタを介して DS レジスタにセグメント値セット
        MOV     DX,BDATA+8ØH        透明データ補正
        MOV     CX,8                ループ回数セット
BPLP:   ;Bullet Put Loop
        vmovsb  BLUE                セグメント値 BLUE でマクロ展開
        vmovsb  RED                 セグメント値 RED でマクロ展開
        vmovsb  GREEN               セグメント値 GREEN でマクロ展開
        ADD     DX,4                次ライン用データ位置を求める
        ADD     BX,HLEN             次ライン VRAM アドレスを求める
        LOOP    BPLP                CX 回ループする
        POP     DS                  DS レジスタ値をスタックから復元
        POP     SI                  SI レジスタ値をスタックから復元
        POP     DX                  DX レジスタ値をスタックから復元
        RET                         リターン
;
;****** List 2-7-N ******
;
KEY:    ;KEY scan                   キー・スキャン
        MOV     AX,Ø4Ø8H            キーコード・グループ番号セット
        INT     18H                 キーグループ8のデータの入力（テンキー4の情報）
        MOV     DL,AH               入力したデータの保存
        MOV     AX,Ø4Ø9H            キーコード・グループ番号セット
        INT     18H                 キーグループ9のデータの入力（テンキー6の情報）
        SHR     AH,1                テンキー6に対応するビットをキャリー・フラグへ
        RCR     DL,1                キャリー・フラグから DL へビット情報を取り込む
        AND     DL,ØAØH             不必要なビット・データを0にする
```

```
        MOV     DH,ZENKAI               前回のキー入力値をDHレジスタへ
        XOR     DH,DL                   前回のキー入力値と新しい値とのXORをとる
        JE      ONAJI                   前回と新しい値が同じならONAJIへ
        MOV     ZENKAI,DL               新しい値を次のチェック用に保存
        AND     DH,DL                   キーが押されたか否かの判断用
        JE      KEYOFF                  キーが離された場合にはKEYOFFへ
        SHL     DH,1                    [6]が押されたか否かの判断用
        JB      KEY6                    [6]が押された場合KEY6へ
        JMP     KEY4                    [4]が押されたときの処理を行う
ONAJI:  ;ONAJI
        MOV     AL,KPKYDT               前回の方向データを取り出す
        CMP     AL,1                    前回の方向データは右である
        JE      KEY6                    [6]の処理へ
        CMP     AL,5                    前回の方向データは左である
        JE      KEY4                    [4]の処理へ
        JMP     NOMOVE                  移動無し
KEYOFF: ;KEY OFF                        ―― キーが離された場合の処理をする
        SHL     DL,1                    [6]が押されているか否かの判断用
        JB      KEY6                    [6]が押されている場合KEY6へ
        SHL     DL,1                    [4]が押されているか否かの判断用
        SHL     DL,1
        JB      KEY4                    [4]が押されている場合KEY4へ
        JMP     NOMOVE                  移動無し
KEY4:   ;KEY  4                         ―― 左方向処理
        MOV     AL,LEND                 ALに左端座標
        SUB     AL,CL                   座標値の確認用
        JE      KEYRET                  座標が左端であるときKEYRETへ
        MOV     AL,5                    移動方向を5(左)とする
        JMP     KEYRET                  KEYRETへジャンプ
KEY6:   ;KEY  6                         ―― 右方向処理
        MOV     AL,REND                 ALに右端座標
        SUB     AL,CL                   座標値の確認用
        JE      KEYRET                  座標が右端であるときKEYRETへ
        MOV     AL,1                    移動方向を1(右)とする
        JMP     KEYRET                  KEYRETへジャンプ
NOMOVE: ;Not MOVE
        MOV     AL,Ø                    移動方向を0(移動無し)とする
        AND     AL,AL                   フラグ・クリア
KEYRET: ;KEY RET
        MOV     KPKYDT,AL               移動方向データを保存する
        RET                             リターン

ZENKAI  db      Ø                       前回のキー入力値,保存用
KPKYDT  db      Ø                       移動方向データ保存用
;
MYMOVE: ;MY MOVE                        ―― 主人公の移動
        MOV     CX,word ptr MYLOC       CX←現在いる座標
        CALL    KEY                     キー入力により移動方向を求める
        JE      SKPCLS                  方向=0ならばSKPCLSへ
        CALL    MVCLS                   移動方向別消去,CXには次座標
        MOV     word ptr MYLOC,CX       移動後座標を保存
```

```
SKPCLS: ;SKiP CLS
        XOR     AL,AL                          AL ← 0 …… 主人公のパターン番号
        CALL    DISP                           座標(CL, CH)にパターン AL を表示
        RET                                    リターン
;
BULSTY  equ     46                             BULlet STart Y   …… 弾発射の Y 座標
BULVAL  equ     6                              BULlet VALue     …… 弾の総数
BDATA   equ     2ØØH*6                         Bullet DATA      …… 弾のパターン・データのアドレス
;
SSKCK:  ;Space & Shift Key Check
        LEA     BX,OLDKEY                      前回押されたか否かのキーワークエリア
        MOV     AX,4Ø6H                        キーコードグループ番号セット
        INT     18H                            キーデータ入力
        TEST    AH,1ØH                         SPACE のチェック
        JNE     PUSHKY                         SPACE が押されたら PUSHKY へ
        MOV     AX,4ØEH                        キーコードグループ番号セット
        INT     18H                            キーデータ入力
        TEST    AH,1                           SHIFT キーのチェック
        JNE     PUSHKY                         SHIFT キーが押されたら PUSHKY へ
        MOV     byte ptr [BX],ØFFH             [BX]← 0FFH
        RET                                    リターン
PUSHKY: ;PUSH KeY
        MOV     CH,[BX]                        CH ← DS : [BX]
        INC     CH                             CH ← CH+1
        JE      $+3                            CH=0 でなければリターン
        RET                                    リターン
        MOV     [BX],AL                        キーデータ保存
        MOV     CL,Ø                           CL ← 0
        CALL    BAPOS                          弾の発射準備をするため
        MOV     CL,3                           CL ← 3
BAPOS:  ;Bullet Appear POSsibility
        MOV     BX,offset BULWOK               BX ←弾のワークエリア先頭アドレス
        MOV     CH,BULVAL                      CH ←弾の総数
BALOOP: ;Bullet Appear LOOP
        MOV     AL,[BX]                        AL ← DS : [BX] …… 弾の出現フラグ
        OR      AL,AL                          AL=0 かどうかのチェック
        JE      BULAP                          AL=0 であれば BULAP
        ADD     BX,3                           BX ← BX+3 …… 次の弾のワークエリア
        DEC     CH                             CH ← CH−1
        JNE     BALOOP                         CH=0 でなければ BALOOP
        RET                                    リターン
BULAP:  ;BULlet APpear
        MOV     byte ptr [BX],1                [BX]← 1 …… 弾の出現フラグ
        INC     BX                             BX ← BX+1
        MOV     AL,MYLOC                       AL ←主人公の X 座標
        ADD     AL,CL                          AL ← AL+CL
        MOV     [BX],AL                        DS : [BX]← AL …… 弾の X 座標
        INC     BX                             BX ← BX+1
        MOV     byte ptr [BX],BULSTY           [BX]←弾発射時の Y 座標
        RET                                    リターン
```

```
;
ABMOVE: ;All Bullet MOVE
        MOV     BX,offset BULWOK        BX←弾のワークエリア先頭アドレス
        MOV     CH,BULVAL               CH←弾の総数
BMLOOP: ;Bullet Move LOOP
        MOV     AL,[BX]                 AL←DS:[BX]…… 出現フラグ
        PUSH    BX                      BXの値をスタックへ
        OR      AL,AL                   AL=0かどうかのチェック
        JE      $+5                     AL=0であれば次命令をスキップ
        CALL    BMOVE                   BMOVEをコール
        POP     BX                      BXの値をスタックから復元
        ADD     BX,3                    BX←BX+3
        DEC     CH                      CH←CH-1
        JNE     BMLOOP                  CH=0でなければBMLOOP
        RET                             リターン
;
BMOVE:  ;Bullet MOVE
        PUSH    CX                      CXの値をスタックへ退避
        INC     BX                      BX←BX+1
        MOV     CX,[BX]                 CX←弾のX,Y座標
        INC     BX                      BX←BX+1
        PUSH    BX                      BXの値をスタックへ退避
        MOV     BX,1Ø8H                 BX←消去のサイズ
        CALL    CLPTXY                  (CL,CH)より,サイズBXで消去
        POP     BX                      BXの値をスタックから復元
        MOV     AL,[BX]                 AL←DS:[BX]
        OR      AL,AL                   AL=0かどうかのチェック
        JE      BLDSAP                  A=0であればBLDSAP
        DEC     byte ptr [BX]           弾のY座標更新
        MOV     CH,[BX]                 CH←弾のY座標
        DEC     BX                      BX←BX-1
        MOV     CL,[BX]                 CL←弾のX座標
        CALL    BLPUT                   (CL,CH)に弾を表示
        POP     CX                      スタックからCXレジスタ値復元
        RET                             リターン
;
BLDSAP: ;BuLlet DiSAPpear
        SUB     BX,2                    BX←BX-2
        MOV     byte ptr [BX],Ø         [BX]←0
        POP     CX                      CXの値をスタックから復元
        RET                             リターン
;
OLDKEY  db      Ø                       OLD pressed KEY …… 前回のキーデータ
MYLOC   db      2 dup (Ø)               MY LOCation     …… 主人公の座標
BULWOK  db      18 dup (Ø)              BULlet WOrK area …… 弾のワークエリア

        include LIST2-6.ASM             LIST2-6.ASMを取り込む
```

テスト2-7 テスト・プログラム(TEST 2-7.ASM)

```
;****** List 2-7-T ******

CODE    segment                         命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,ES:PTNSEG,SS:STSEG

print   macro   string                  文字列出力マクロ定義スタート
        LEA     DX,string               string に対するオフセット・アドレスを得る
        MOV     AH,9                    文字列出力コマンドセット
        INT     21H                     ファンクションコール
        endm                            マクロ定義終了
;
PTEST:  ;Program TEST
        CLD                             ディレクション・フラグ・リセット
        MOV     AX,CS                   AX ← CS
        MOV     DS,AX                   AX レジスタを介して DS に CS を格納
        print   CLEAR                   テキスト画面クリア
        CALL    DATLD                   パターン・データ・ロード
        CALL    GINIT                   グラフィック・システム・クリア
        MOV     BX,2E1AH                BX ←主人公の初期出現座標
        MOV     word ptr MYLOC,BX
        MOV     BX,offset BULWOK        BX ←弾のワークエリアの先頭アドレス
        MOV     CX,BULVAL               CX ←弾の総数
TL:     ;Test Loop
        MOV     byte ptr [BX],Ø         弾の出現フラグを0にする
        ADD     BX,3                    BX ← BX+3 …… 次の弾のワークエリア
        LOOP    TL                      弾の数だけループする
;
TMAIN:  ;Test MAIN loop
        MOV     AX,ØCØØH                キーボード・バッファ・クリア
        INT     21H
        CALL    MYMOVE                  主人公の移動
        CALL    SSKCK                   [SPACE], [SHIFT] をチェックするため
        CALL    ABMOVE                  すべての弾をチェックするため
        MOV     CX,1ØØØH                ウェイト用
TESWAIT:;Test WAIT
        NOP                             ウェイト用
        LOOP    TESWAIT                 CX 回ループ
        MOV     AX,Ø4ØØH                キーコード・グループ番号0入力用
        INT     18H                     キーデータ入力
        ROR     AH,1                    ESC データをキャリーへ
        JNB     TMAIN                   [ESC] が押されていなければ TMAIN へ
        print   CSRON                   カーソル・オン
        MOV     AX,ØCØØH                キーボード・バッファ・クリア
        INT     21H
        MOV     AL,Ø                    リターン・コード←ノーマル(0)
        MOV     AH,Ø4CH                 プロセスの終了
        INT     21H                     ファンクションコール
```

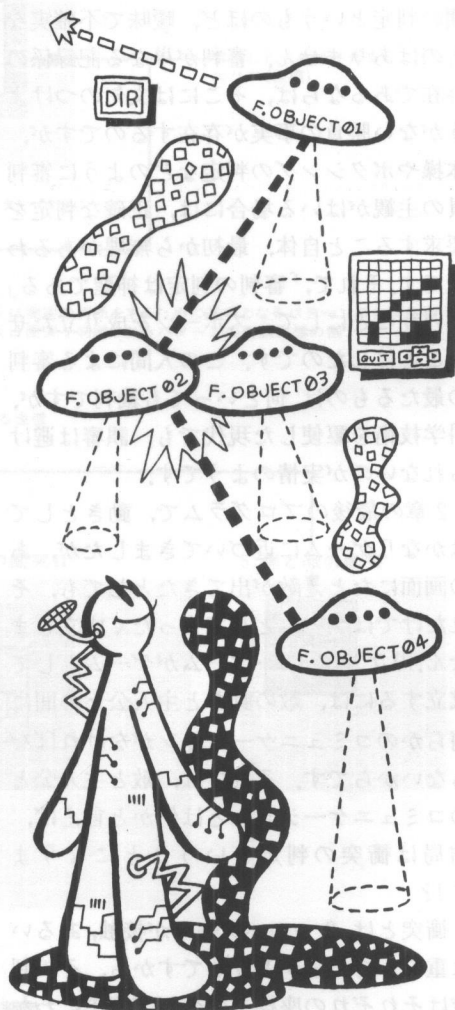
```
;
lodinf   struc                                    ロード情報構造体定義
CMD      db       Ø                               ロード・コマンド
PASS     db       "           "                   パス格納領域(11文字分)
ENDSIN   db       Ø                               パス・エンド・コード
LDADR    dw       Ø                               ロード・アドレス
LDSEG    dw       Ø                               ロード・セグメント
lodinf   ends                                     構造体定義終了
;
LODTBL   lodinf <1,"PTNDAT1.DAT" ,Ø,Ø,PTNSEG>
         lodinf < >
;
         include LIST2-7.ASM                      LIST2-7.ASMを取り込む
```

3章

●衝突と得点計算

●人間という動物は，何にでも優劣をつけたがるもので，一般社会では給料や肩書によって差をつけていますし，学生社会においては試験による順位があります．その結果，社会での自分位置付けなるものを自分自身でサトリ？「まァこんなところでいいャ」なんてあきらめと，中流意識が入り混じると最悪．

●とくに，コンピュータ・ゲームに中流意識やあきらめは通用しません．ハイゲームを作りたい人は，スペースバーが折れようとゲームを続け，そのゲームをキワメなくてはなりません．それもいやな人は，自分でオリジナルなゲームを作ってしまいましょう．自分の作ったゲームについては，すべて知っているワケだから……！

●というわけで，本書では，２章で作った表示や移動ルーチンに，衝突判定と得点計算を付け加え，ミニ・シューティングゲームを作ります．このシューティングゲームは，マシン語の勉強用にスーパー・サブセットとなっていますので，「なァんだ，おもしろくないなァー」と言うのは，５章，６章のゲームを経験した後にしてください．

1. 衝突の判定…ゲーム座標を用いる

スポーツにもいろいろな種類がありますが，審判の手によって勝敗が付けられるものがたくさんあります．しかし，およそ人間の判定というものほど，曖昧で不確実なものはありません．審判が単なる記録係の存在であるならば，そこには文句のつけようがない勝負の事実が存在するのですが，体操やボクシングの判定などのように審判員の主観がはいる場合には，正確な判定を要求すること自体，最初から無理があるわけです．それで，「審判の判定は神聖である」ということにして，スポーツを成り立たせることにしたのです．この人間による審判の最たるものは，何といっても裁判ですが，科学技術を駆使した現代でも，誤審は避けられないのが実情のようです．

2章の最後のプログラムで，動きとしてはかなりゲームに近づいてきましたが，あの画面にたとえ敵が出てきたとしても，それだけではゲームとしてまったく成立しません．リアルタイム・ゲームがゲームとして成立するには，敵の動きと主人公との間に何らかのコミュニケーションがなければならないからです．それでは，敵と主人公とのコミュニケーションとは何かと言えば，結局は衝突の判定ということになります!?

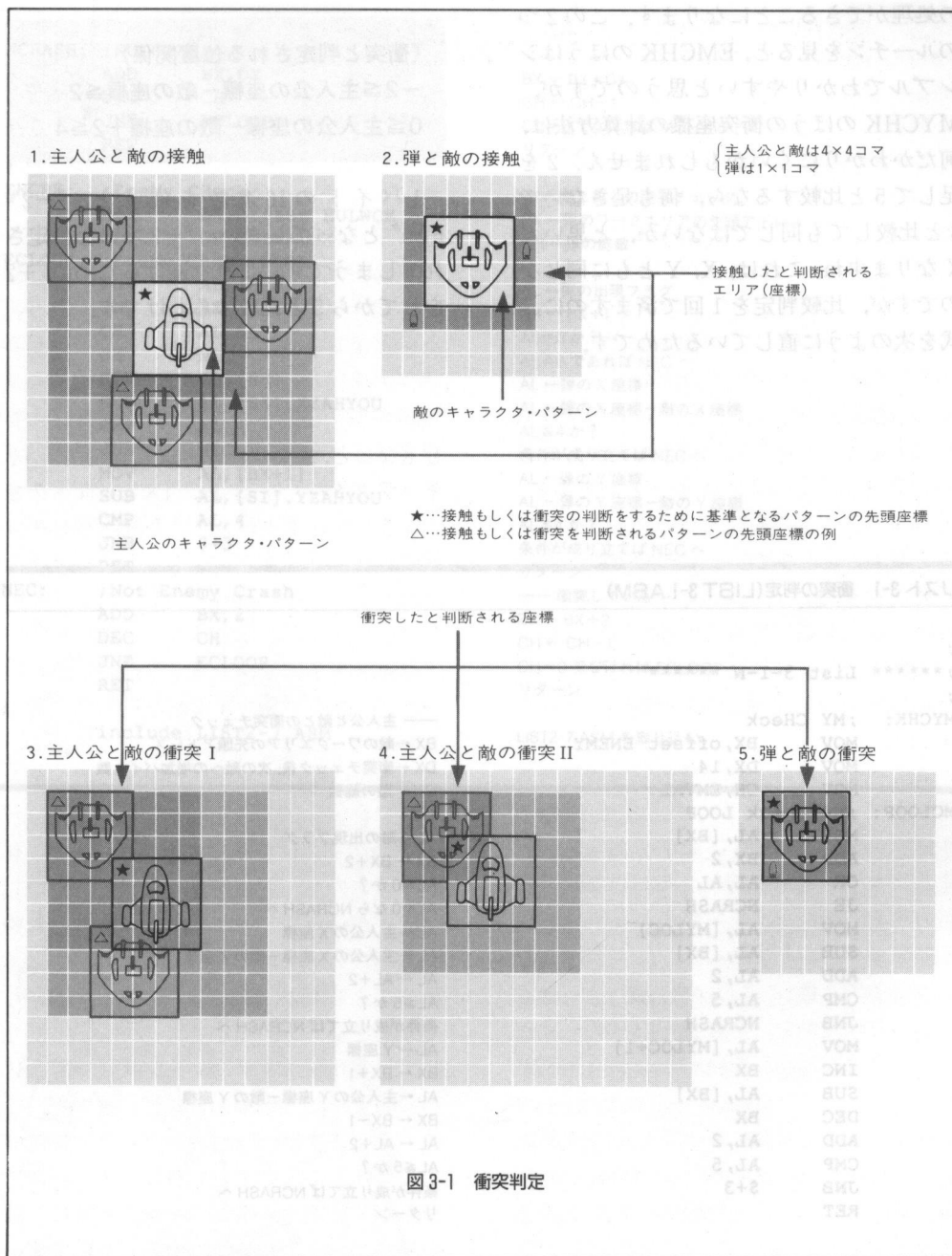
衝突とは，2つのパターンが接触，あるいは重なっているかどうかですから，その判定はそれぞれの座標を調べれば簡単にわかるわけです．これを，具体的に示すと**図3-1**のようになります．

この図で言う接触の判定とは，2つのパターンが出会ったときに互いにハジキ合うというような場合(ピンボール・ゲームなど)に用いられます．いっぽう，衝突というのは2つのパターンがある程度，重なった場合をいいます．どちらにしても判定の基準は図3-1で示されるようにパターン左上(先頭座標)の位置関係になります．

衝突Iの判定は，敵と主人公が1コマでも重なると，衝突したとみなす判定法です．キャラクタ・パターンは，4×4ブロックで構成されていますが，その全部にパターンが描かれているのではなく，空白部分も含んでいます．そのため1コマ重なるだけで衝突したと判定されたのでは，描かれた図柄自体は重なっていない場合があり，ゲームとしてはキビシ過ぎるといえます．そこで衝突IIのようにパターンの1/8(2コマ)以上が重なったとき，初めて衝突したと判定することにしています．なお，弾と敵の衝突についてはややキビシク，敵のパターン(4×4ブロック)内に弾があれば衝突と判定しています．ゲームをするあなたにとってやや有利であると言えます．

リスト3-1のMYCHKルーチンが，敵と主人公との衝突判定をするルーチンで，EMCHKルーチンは敵と弾との衝突判定をしています．判定の結果はどちらもキャリーフラグで返していますから，メインルーチン(**リスト3-5**のTESTルーチン)ではこの判定プログラムの後で，キャリーフラグによる条件分岐をさせれば，衝突後

1.主人公と敵の接触
2.弾と敵の接触
主人公と敵は4×4コマ
弾は1×1コマ
接触したと判断される
エリア(座標)
敵のキャラクタ・パターン
主人公のキャラクタ・パターン
★…接触もしくは衝突の判断をするために基準となるパターンの先頭座標
△…接触もしくは衝突を判断されるパターンの先頭座標の例
衝突したと判断される座標
3.主人公と敵の衝突 I
4.主人公と敵の衝突 II
5.弾と敵の衝突

図3-1　衝突判定

の処理ができることになります．この2つのルーチンを見ると，EMCHK のほうはシンプルでわかりやすいと思うのですが，MYCHK のほうの衝突座標の計算方法は，何だかわかりにくいかもしれません．2を足して5と比較するなら，何も足さないで3と比較しても同じではないか，と思いたくなりますね．これは，X，Yともに同じなのですが，比較判定を1回で済ますのに，式を次のように直しているためです．

《衝突と判定される位置関係》
−2≦主人公の座標−敵の座標≦2
0≦主人公の座標−敵の座標＋2≦4

1バイトの16進数を使うと，〝−2＝FEH″となってしまい，〝−2＞3″と判定されてしまうです．これを避けるために，＋2をしてから5と比較するわけです．

リスト3-1　衝突の判定(LIST 3-1.ASM)

```
;
;****** List 3-1-N ******
;
MYCHK:   ;MY CHeck                    ── 主人公と敵との衝突チェック
         MOV     BX,offset ENEMY      BX←敵のワークエリアの先頭アドレス
         MOV     DX,14                DX←衝突チェック後，次の敵への増加バイト数
         MOV     CH,EMVAL             CH←敵の総数
MCLOOP:  ;My Check LOOP
         MOV     AL,[BX]              AL←敵の出現フラグ
         ADD     BX,2                 BX←BX+2
         OR      AL,AL                AL=0か？
         JE      NCRASH               AL=0ならNCRASHへ
         MOV     AL,[MYLOC]           AL←主人公のX座標
         SUB     AL,[BX]              AL←主人公のX座標−敵のX座標
         ADD     AL,2                 AL←AL+2
         CMP     AL,5                 AL≦5か？
         JNB     NCRASH               条件が成り立てばNCRASHへ
         MOV     AL,[MYLOC+1]         AL←Y座標
         INC     BX                   BX←BX+1
         SUB     AL,[BX]              AL←主人公のY座標−敵のY座標
         DEC     BX                   BX←BX−1
         ADD     AL,2                 AL←AL+2
         CMP     AL,5                 AL≦5か？
         JNB     $+3                  条件が成り立てばNCRASHへ
         RET                          リターン
```

```
NCRASH: ;No CRASH
        ADD     BX,DX                   BX ← BX+DX
        DEC     CH                      CH ← CH−1
        JNE     MCLOOP                  CH=0でなければMCLOOPへ
        RET                             リターン
;
EMCHK:  ;EneMy CHeck                    ── 敵と弾との衝突チェック
        MOV     BX,offset BULWOK        BX ← 弾のワークエリアの先頭アドレス
        MOV     CH,BULVAL               CH ← 弾の総数
ECLOOP: ;EneMy Check LOOP
        MOV     AL,[BX]                 AL ← 弾の出現フラグ
        INC     BX                      BX ← BX+1
        OR      AL,AL                   AL=0か？
        JE      NEC                     AL=0であればNECへ
        MOV     AL,[BX]                 AL ← 弾のX座標
        SUB     AL,[SI].XZAHYOU         AL ← 弾のX座標−敵のX座標
        CMP     AL,4                    AL≦4か？
        JNB     NEC                     条件が成り立てばNECへ
        MOV     AL,[BX+1]               AL ← 弾のY座標
        SUB     AL,[SI].YZAHYOU         AL ← 弾のY座標−敵のY座標
        CMP     AL,4                    AL≦4か？
        JNB     $+3                     条件が成り立てばNECへ
        RET                             リターン
NEC:    ;Not Enemy Crash                ── 衝突している
        ADD     BX,2                    BX ← BX+2
        DEC     CH                      CH ← CH−1
        JNE     ECLOOP                  CH=0でなければECLOOPへ
        RET                             リターン
;
        include LIST2-7.ASM             LIST2-7.ASMを取り込む
```

2. 数字…文字と数字パターンの作成

衝突のチェックが済めば，次は後処理をしなければなりません．つまり，裁判でいうなら刑の執行となるか，無罪放免となるかですが，ゲームでは，敵が弾に当たって爆発するとか，点数をアップするとか，主人公がやられたらゲーム・オーバーになるとか……，ということになります．ゲームでは，死んでもすぐに生き返れるので，死への恐怖などというものはだれも感じないと思います．しかし，元来人間にとってこの恐怖はとても大きなものだったのです．そのために生まれたのが宗教であり，キリスト様も，お釈迦様も，アラーの神も……，すべてこの死への恐怖をとり除いてくれる(?)という点で一致しているのです．そういう意味においては，コンピュータ・ゲームは正に時代の最先端を行く宗教といえよう……!?

ここでは，そのような恐怖(?)処理の準備として，数字や文字を表示するためのルーチンを作成します．数字や文字を表示するルーチンといっても，基本的な画面表示の考え方は，パターン表示ルーチンととくに変わりはありません．ただ，文字パターンのサイズ，色(ここでは白に統一)，連続表示などの点から，これまでのパターン表示プログラムに少し変化があるという程度です．

そこで，まず必要になってくるのは数字や文字のグラフィック・データです．これがないと表示プログラムが正しいかどうかの実験もできませんから，ここはめんどうでもすべての数字，文字用グラフィック・データをパターン・エディタで作ってしまいます．パターン・サイズは，16×16ドットとなります．また，連続したときに上下左右が触れないよう目次の後ろに用意した数字・文字パターンのように最初から一回り小さく作ります．細かいことを言えば，下部のスペースはわざわざデータで持つ必要はないのですが，ここではデータの管理をわかりやすくするために，あえてデータとしてあります．

グラフィック・データは1画面分しか必要としないのですが，パターンエディタでは透明，青，赤，緑の4つ分のデータがあります．もっとも，この透明に関してのデータは削除することができます．しかし，将来カラフルな文字をデザインしたり，重ね合わせたりする場合もあるでしょうから，ここではそのままパターン・エディタのデータを使うことにします．この場合，パターンサイズはキャラクタ・パターンと異なりますので，別なファイルとして管理したほうがいいでしょう．また，それぞれの文字データは80Hバイト(2×10H×4=80H)ごとに格納されていることに注意してください．

データの転送先で，数字の"9"と文字"A"との間に"□"が挿入されていますが，このパターンは，パターン・エディタで作らなくてもけっこうです．これは，スペースすなわち1文字分の空送りをするときに，消去用のデータとして使っているからです．

そして，すべての数字・文字データがそろったら，SYMDEBなどで1つのファイルにまとめておいてください．ここでは，このまとめたファイル名をMOJI.DATとしました．

さて，リスト3-2の内容はこれら1文字分の表示，画面全部の消去，連続文字の表示の3部から成っています．このプログラムでは，これまでパターンの表示アドレスを求めるために使っていたパターン番号からデータ・アドレスを計算するルーチン(PDADR)が使われていません．その代わりにSEEKLDという別の変換ルーチンが出てきています．これは，とくに意味のあることではなく，PDADRルーチンではパターンサイズがバラバラでも利用できる例として用い，また今回のようにデータ長が一定の場合には，ポインタをわざわざ確保する必要がないので，別のルーチンにしただけのことです．したがって，数字・文字にはこれまでのパターン番号とは別に，次のようなパターン番号が付いていると解釈できます．

パターン番号00〜09	：数字の0〜9
パターン番号10	：スペース
パターン番号11〜36	：文字のA〜Z

ここで工夫を凝らしているのは数字，文字の連続表示ルーチンで，連続表示データとしてASCIIコードが使えるという点です．これは，数字表示の場合はあまりメリットがありませんが，文字を表すときには表示したい文字をダブルクォート(”)で囲むだけで，連続表示でき文字表示が大幅に楽になるからです．ASCIIコードから前述のパターン番号に直す分だけ，プログラムとしてはほんの少し長くなりますが，MESSの内容を文字データにしてアセンブルすると，パターン番号でデータを作ることがいかにめんどうか，すぐわかると思います．連続表示のエンド・サインは0となっているので，これは間違っても「”」で囲まないようにしてください．

もう1つのルーチン，画面クリアについてはGDCやEGCを使うともっと効率のいい高速なルーチンができますが，ここではPC-9801シリーズに共通で使えるようにするため，基本的な方法で実現しています．

リスト 3-2 文字の表示(LIST 3-2.ASM)

```
;****** LIST3-2 *******
DISPLE: ;DISPlay LEtter              ―― 文字・数字の表示
        PUSH    SI                   SI レジスタ値をスタックへ退避
        PUSH    DS                   DS レジスタ値をスタックへ退避
        CALL    XYADR                表示アドレスを求めるため
        CALL    SEEKLD               データのアドレスを求めるため
        MOV     SI,AX                SI ← AX
BOXL:   ;BOX of Letter               ―― 1文字の表示
        MOV     DI,DISAD             DI ←表示アドレス
        MOV     CX,16                CX ←縦のドット数
        MOV     AX,PTNSEG            文字列パターンの格納されたセグメント・アドレス
        MOV     DS,AX                AX レジスタを介して DS にセグメント値を格納
LLOOP:  ;Letter LOOP
        MOV     AX,BLUE              AX ←ブルー面セグメント値
        MOV     ES,AX                AX を介して ES にセグメント値を格納
        MOV     DX,[SI]              DX ← DS : [SI]
        AND     ES:[DI],DX           ES : [DI]← ES : [DI] AND DX
        MOV     AX,[SI+20H]          AX ← DS : [SI+20H]
        OR      ES:[DI],AX           ES : [DI]← ES : [DI] OR AX
        MOV     AX,RED               AX ←レッド面セグメント値
        MOV     ES,AX                AX を介して ES にセグメント値を格納
        AND     ES:[DI],DX           ES : [DI]← ES : [DI] AND DX
        MOV     AX,[SI+40H]          AX ← DS : [SI+40H]
        OR      ES:[DI],AX           ES : [DI]← ES : [DI] OR AX
        MOV     AX,GREEN             AX ←グリーン面セグメント値
        MOV     ES,AX                AX を介して ES にセグメント値を格納
        AND     ES:[DI],DX           ES : [DI]← ES : [DI] AND DX
        MOV     AX,[SI+60H]          AX ← DS : [SI+60H]
        OR      ES:[DI],AX           ES : [DI]← ES : [DI] OR AX
        ADD     DI,HLEN              表示アドレスを次ラインにする
        ADD     SI,2                 SI ← SI+2
        LOOP    LLOOP                CX 回ループ
        POP     DS                   スタックから DS レジスタ値を復元
        POP     SI                   スタックから SI レジスタ値を復元
        RET                          リターン
;
VTOP    equ     0                    VRAM の先頭アドレス
;
vstosw  macro   vramseg              VRAM への STOSW マクロ定義
        MOV     CX,vramseg           CX ← vramseg
        MOV     ES,CX                CX レジスタを介して ES へセグメント値セット
        MOV     CX,40                1 ライン分のループ回数セット
        MOV     DI,BP                VRAM アドレス初期化
        REP     STOSW                ES : [DI]← AX
        endm                         マクロ定義終了
```

```
CLS:    ;CLear Screen                           ── 画面の高速消去
        MOV     BP,VTOP                         BP ← VRAMの先頭アドレス
        MOV     DX,4ØØ                          DX ← 400ライン分のループ回数セット
        XOR     AX,AX                           AX ← 0
CLSLP1: ;CLear Screen LooP 1
        vstosw  BLUE                            セグメント値BLUEでマクロ展開
        vstosw  RED                             セグメント値REDでマクロ展開
        vstosw  GREEN                           セグメント値GREENでマクロ展開
        MOV     BP,DI                           BP ← DI
        DEC     DX                              DX ← DX-1
        JNE     CLSLP1                          DX=0でなければCLSLP1へ
        RET                                     リターン

SEEKLD: ;SEEK Letter Data
        MOV     AH,Ø                            AH ← 0
        XCHG    AL,AH                           AX ←文字サーチ・コード番号×100H
        SHR     AX,1                            AX ← AX÷2 …… コード番号×80Hに相当
        ADD     AX,2ØØØH                        AX ← AX+2000H
        RET                                     リターン
;
MSGPRN: ;MeSsaGe PRiNt
        MOV     AL,[BX]                         AL ← DS:[BX]
        OR      AL,AL                           AL=0か?
        JNE     $+3                             0に等しければリターン
        RET                                     リターン
        CMP     AL,' '                          AL=' 'か?
        JNE     MSG2                            等しくなければMSG2へ
        MOV     AL,'Ø'+1Ø                       AL ← 30H+10 ……空白を表す
MSG2:   ;MeSsaGe print-2
        SUB     AL,'Ø'                          AL ← AL-30H
        CMP     AL,11                           AL<11か?
        JB      MSG1                            AL<11であればMSG1へ
        SUB     AL,6                            AL≧11であれば-6の補正
MSG1:   ;MeSsaGe print-1
        PUSH    CX                              スタックへCXレジスタ値を退避
        PUSH    BX                              スタックへBXレジスタ値を退避
        CALL    DISPLE                          (CL,CH)よりALを表示 …… 文字・数字・空白
        POP     BX                              スタックからBXレジスタ値を復元
        POP     CX                              スタックからCXレジスタ値を復元
        ADD     CL,2                            CL ← CL+2
        INC     BX                              BX ← BX+1
        JMP     MSGPRN                          MSGPRNへジャンプ

        include LIST3-1.ASM                     LIST3-1.ASMを取り込む
```

テスト3-2 テスト・プログラム(TEST3-2.ASM)

```
;****** List 3-2-T ******
CODE    segment                                     命令の置かれているセグメントの始まり
        assume CS:CODE,DS:CODE,ES:PTNSEG,SS:STSEG
print   macro   string                              文字列出力マクロの定義
        LEA     DX,string                           マクロパラメータのオフセットをDXへ
        MOV     AH,9                                ファンクションコール番号9 …… 文字列出力
        INT     21H                                 ファンクションコール
        endm                                        マクロ定義終了
;
PTEST:  ;Proglam TEST
        CLD                                         ディレクション・フラグのリセット
        MOV     AX,CS                               AX ← CS
        MOV     DS,AX                               AXレジスタを介してDSにCSを格納
        print   CLEAR                               テキスト画面クリア
        CALL    DATLD                               パターン・データ・ロード
        CALL    GINIT                               グラフィック・システム初期化
        CALL    CLS                                 グラフィック画面クリア
        MOV     CX,1Ø1ØH                            CX ←表示スタート座標
        MOV     BX,offset MESS                      BX ←文字列データ・ポインタ
        CALL    MSGPRN                              (CL,CH)より文字列を表示するため
        print   CSRON                               カーソル・オン
        MOV     AL,Ø                                リターン・コード←ノーマル
        MOV     AH,Ø4CH                             AH ←コマンド・セット
        INT     21H                                 ファンクションコール

        ;MESSage
MESS    db      "Ø123456789",Ø

lodinf  struc                                       ロード情報構造体定義
CMD     db      Ø                                   ロード・コマンド
PASS    db      "           "                       パス格納領域 (11文字分)
ENDSIN  db      Ø                                   パス・エンド・コード
LDADR   dw      Ø                                   ロード・アドレス
LDSEG   dw      Ø                                   ロード・セグメント
lodinf  ends                                        構造体定義終了
;
LODTBL  lodinf <1,"PTNDAT1.DAT"  ,Ø,Ø,PTNSEG>
        lodinf <1,"MOJI.DAT"     ,Ø,2ØØØH,PTNSEG >
        lodinf < >
;
        include  LIST3-2.ASM                        LIST3-2.ASMを取り込む
```

3. 計算…得点の計算と表示　その1

数字や文字を自由に画面に表示できるようになったからといって，すぐに得点の表示が可能になったのではありません．得点を画面に表示するには，まだ重要な問題が残っています．それは，コンピュータが計算するのは16進数でありながら，画面に表示するときは10進数であるという点です．つまり，内部では16進数で計算をしていても，人間には10進数表記でないと理解されないということです．たとえ，マシン語でプログラムを組むことができるような人でも，16進数より10進数のほうがわかりやすいのは当然のことです．このあたりのギャップが，人間の頭脳とコンピュータとの基本的な構造上の違いであるといえます．

この責任のすべては，人間を創造した神様にあります．もしも，人間の指が片手に8本ずつあったならば，最初からすべて16進数の世界になっていたはずです．おそらく神様も人間がこのような奇怪な機械を創造するとは夢にも思わなかったのでしょう．コンピュータの出現にいちばん驚いたのは，そういう意味では人間を創造した神様であったかもしれません．しかし，今後は人工知能を持ったコンピュータがさらに知恵を持った何かを創造する……というようになると，コンピュータにとっての神である人間が，その違いに驚くというときが来るかもしれません．

ここでは，16進数の数字データを10進数の連続文字(数字)列に変換することで，この問題を解決していきます．0から9までの数字の列になれば，作成したばかりの文字，数字連続表示ルーチンを使って画面に出力できるようになるのです．

マシン語で直接計算できる数字は0～FFFFHまでですから，スコア専用のワークエリアとして，2バイトのメモリを用意する必要があります．これは，10進数で言うと0～65535にしかなりませんが，実際にはスコア表示の際は，ダミーとして最後に00を付けておくことにより0～6553500という高い点数にすることができます．これだけの点数があれば，表示スコア不足になることはまずありません．ダミーとわかっていても，この00がないと不満があるというのは，ゲームの世界も相当にインフレが進んでいるからでしょう．

リスト3-3は，このような2バイトの16進数からなるスコアを，DXで示される数字と加算した上で，10進数の文字列に変換し，指定位置から表示するというプログラムです．

リストのコメントを見れば明らかなように16進数から10進数への変換は求める桁ごとに割り算をして，その桁の数字を出しています．この変換計算の考え方は，次のように10進数の数字でやってみると理解しやすくなります．

例 65535(FFFF$_H$)の各桁の値を求める
1. 10000(2710$_H$)で65535(FFFF$_H$)を割る
商 ……6 10000の位
余り…… 5535(159F$_H$)
2. 1000(3E8$_H$)で5535(159F$_H$)を割る
商 ……5 1000の位
余り…… 535(217$_H$)
3. 100(64$_H$)で535(217$_H$)を割る
商 ……5 100の位
余り…… 35(23$_H$)
4. 10(0A$_H$)で35(23$_H$)を割る
商 ……3 10の位
余り…… 5 1の位

結局,割る数も割られる数も16進数でやれば,各桁の値は同じように求められるのです.そして,この各桁の値に30$_H$を加えることにより,この数字がASCIIコードとなります.これらを,順次指定のメモリに格納することによって,それらはそのまま連続表示用のデータとなるので,データ終了のサイン00を最初から1の位の次のメモリに入れておけばOKです.

なお,ここでの割り算では,扱う数値を符号なしの整数と仮定して,div命令を使っています.数値は万の位も含むので,オペランドは必然的に16ビットとなります.また,被除数は最大で65536としましたから,上位レジスタのDXは0としてあります.

では,実際にスコアを表示させるテストをしてみましょう.表示させる内容は次のように指定します.テストとはいえ,ダミーの00までついた立派なものです.

いつものようにテスト・プログラムを走らせると,スコアが100点(実際は1点)ずつアップしていくはずです.といっても,あまり高速で1/100秒計時のストップ・ウォッチのように見えるかもしれません.このテストには,終わりがないので適当に ESC を押して止めてください.

もし,画面数などで2桁の数字を表示したい場合は,連続表示データのスタートを現在のF10000からF10に変更すれば,2桁の表示に変更することができます.ところで,せっかく理解してきたこの得点表示プログラムなのですが,実は本書ではこのテストプログラム以外に使われることのない,幻のプログラムになる運命なのです…….

リスト 3-3　得点の計算と表示1(LIST 3-3.ASM)

```
;
;****** List 3-3-N ******
;
tensuu  macro   tf,td              得点表示用データ作成マクロ定義
        XOR     DX,DX              DX←0
        MOV     CX,td              CX←td：割る数セット
        DIV     CX                 DX：AX÷CX
        ADD     AL,30H             AL←商に対するASCII補正
        MOV     tf,AL              データを保存
        MOV     AX,DX              AX←DX：余り
        endm                       マクロ定義終了
;
DSC1:   ;Display SCore-1
        PUSH    CX                 CXレジスタ値をスタックへ退避
        MOV     AX,SCORE           AX←得点
        ADD     AX,DX              得点アップ
        MOV     SCORE,AX           得点の保存
        tensuu  F10000,10000       10000点でtensuuマクロ展開
        tensuu  F1000,1000         1000点でtensuuマクロ展開
        tensuu  F100,100           100点でtensuuマクロ展開
        tensuu  F10,10             10点でtensuuマクロ展開
        ADD     AL,30H             AL←'0'
        MOV     F1,AL              得点の1の位のデータ保存
        POP     CX                 スタックからCXレジスタ値を復元
        MOV     BX,offset F10000   文字列の先頭アドレス
        CALL    MSGPRN             (CL,CH)より文字列を表示
        RET                        リターン
;
F10000  db      0       ;Figure 10000   10000の位の値がASCIIコードではいる
F1000   db      0       ;Figure 1000    1000の位の値がASCIIコードではいる
F100    db      0       ;Figure 100     100の位の値がASCIIコードではいる
F10     db      0       ;Figure 10      10の位の値がASCIIコードではいる
F1      db      0       ;Figure 1       1の位の値がASCIIコードではいる
FEND    db      0       ;Figure END
SCORE   dw      0       ;SCORE

        include LIST3-2.ASM        LIST3-2.ASMを取り込む
```

テスト 3-3　テスト・プログラム(TEST 3-3.ASM)

```
;
;****** List 3-3-T ******
;
CODE    segment                    命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,ES:PTNSEG,SS:STSEG
```

```
print   macro   string                          文字列出力マクロの定義
        LEA     DX,string                       マクロパラメータのオフセットをDXへ
        MOV     AH,9                            ファンクションコール番号9 …… 文字列出力
        INT     21H                             ファンクションコール
        endm                                    マクロ定義終了
;
PTEST:  ;Program TEST
        CLD                                     ディレクション・フラグ・リセット
        MOV     AX,CS                           AX ← CS
        MOV     DS,AX                           AXレジスタを介してDSにCSを格納
        print   CLEAR                           テキスト画面クリア
        CALL    DATLD                           パターン・データ・ロード
        CALL    GINIT                           グラフィック・システム初期化
        CALL    CLS                             グラフィック画面クリア
        XOR     AL,AL                           AL ← 0
        MOV     FEND,AL                         エンド・サイン・セット
        MOV     AL,3ØH                          AL ← 0のASCIIコード
        MOV     F1Ø,AL                          ダミー・スコアをセット
        MOV     F1,AL                           ダミー・スコアをセット
        MOV     BX,Ø                            BX ← 0
        MOV     SCORE,BX                        スコア初期化
        MOV     CX,ØØ4AH                        CX ←ダミースコア「00」の位置
        MOV     BX,offset F1Ø                   点数表示用領域アドレス・セット
        CALL    MSGPRN                          (CL, CH)より文字列を表示する
TLOOP:  ;Test LOOP
        MOV     DX,1                            DX ← 1
        MOV     CX,ØØ4ØH                        スコア表示座標
        CALL    DSC1                            スコアにDXを加算して(CL, CH)より表示
        MOV     AX,Ø4ØØH                        キーコード・グループ番号0入力用
        INT     18H                             キーデータ入力
        ROR     AH,1                            ESCデータをキャリーへ
        JNB     TLOOP                           [ESC]が押されていなければTLOOPへ
        print   CSRON                           カーソル・オン
        MOV     AX,ØCØØH                        }
        INT     21H                             } キーボード・バッファクリア
        MOV     AL,Ø                            リターン・コード←ノーマル
        MOV     AH,Ø4CH                         プロセスの終了コマンド・セット
        INT     21H                             ファンクションコール
;
lodinf  struc                                   ロード情報構造体定義
CMD     db      Ø                               ロード・コマンド
PASS    db      "           "                   パス格納領域(11文字分)
ENDSIN  db      Ø                               パス・エンド・コード
LDADR   dw      Ø                               ロード・アドレス
LDSEG   dw      Ø                               ロード・セグメント
lodinf  ends                                    構造体定義終了
;
LODTBL  lodinf <1,"PTNDAT1.DAT"  ,Ø,Ø,PTNSEG>
        lodinf <1,"MOJI.DAT"     ,Ø,2ØØØH,PTNSEG >
        lodinf < >
;
        include LIST3-3.ASM                     LIST3-3.ASMを取り込む
```

4. BCD&ASCII…得点の計算と表示　その2

普通のゲームであれば，16進数→10進数への換算による得点表示でもまったく支障はありませんし，困ることはありません．しかし，得点を表示するたびにイチイチ換算するというのは，どう考えても合理的な方法であるとは言えません．それに，たとえ不足することがないといっても，数値の上限に最初から制限があるというのもあまり気分のイイものではありません．この2つの不満を，一気に解消するようなウマイ方法はないものでしょうか…….

実は，この16進数と10進数との問題は，ゲームばかりではなくコンピュータと人間がコミュニケートする上で，つねに存在している大きな障害なのです．

たとえば，電卓などは計算がすべてという商品ですから「入力は10進数，内部計算は16進数，表示は10進数」ではたまりません．そこで，16進数を10進数の感覚で使ってしまうのが，BCD(Binary Coded Decimal)＝2進化10進数の考え方なのです．

これは何を意味するかというと，使う側が0から9までの数字だけを使い，16進数を10進数とみなしてしまおうというものです．ですから，計算をしない限りは10進数そのものとまったく同じことなのです．

例
16進数で12＝10進数で12とみなす
16進数で15＝10進数で15とみなす

このように，16進数を10進数と同じものと考えることは，考える人の勝手ということになりますが，ここに計算処理がはいるとそう単純にはいかなくなります．つまり，コンピュータはあくまで16進数しか処理できないからです．そのために，計算をしたときにはかならず16進(2進)から10進への補正処理をする必要がでてきます．

例
12H＋3H＝15H …… 補正 → 15
（そのままで良い）
12H＋9H＝1BH …… 補正 → 21
18H＋8H＝20H …… 補正 → 26
37H－9H＝2EH …… 補正 → 28

この16進(2進)から10進への補正は，計算をするたびにかならずしなければなりません．しかし，実際には，この補正はたった1つの命令DAA(Decimal Adjust for Addition)で解決できるのです．DAAは加算用ですが，もちろん，DAS(Decimal Adjust for Subtraction)という減算用補正命令もあります．なお，演算結果はALにはいるという制限がありますので注意してください．

さて，16進数をあたかも10進数として計算してしまうという欲求が満たされると，次は，なんといっても，ASCIIコードの数値をそのまま演算したいという欲求が出てきてもおかしくはありません．

8086には，ちゃんと，ASCII演算に対する補正も準備されています．しかも，このASCII補正に関しては，加減乗除の四則演算に対して補正が簡単にできるようになっているのです．

数字のASCIIコード	
0＝30_{H}	5＝35_{H}
1＝31_{H}	6＝36_{H}
2＝32_{H}	7＝37_{H}
3＝33_{H}	8＝38_{H}
4＝34_{H}	9＝39_{H}

たとえば，10進数で5＋6＝11をASCIIコードで計算できたとすると35_{H}＋36_{H}＝3131_{H}となるわけです．ところが，マシン語には，ASCIIコードの加算命令はありませんので，計算すると35_{H}＋36_{H}＝$6B_{H}$となります．したがって，〝$6B_{H}$→3131_{H}〟とする補正作業がいることになります．

加算用補正はAAA（ASCII Adjust for Addition）という命令ですが，これは，ALにあらかじめ計算結果が求められているとして，下位ニブル（下位4ビット）に対して10（$0A_{H}$）以上であれば，＋6の補正をし，ALの上位レジスタであるAHに1を加算して，キャリーフラグを立てるという一連の作業をします．また，この命令の実行後はALの上位ニブル（上位4ビット）は0クリアされます．

さて，このAAA命令を使って，さきほどの例を実行すると次のようになります．

```
MOV  AH, 30H  ；AX の上位レジスタにあらかじめ初期値 0 を ASCII コードで
MOV  AL, 35H  ；5 の ASCII コードを AL に格納
ADD  AL, 36H  ；AL＝35H＋36H……6BH が AL の値となる
AAA           ；ASCII 補正(加算用)：AL＝1，AH＝AH＋1，CF＝1 となる
OR   AL, 30H  ；上位ニブルに 3 をセットし最終的に AX＝3131H が求まる
```

この補正を利用して，プログラミングしたのが，リスト 3-4 です．ここでは桁数の上限を 6 桁とし，これにダミーの 00 を付ければ，00～99999900 までの表示ができるということになります．

テスト・プログラムの内容は，リスト 3-3 とまったく同じことをしていますので，実験をしてみてください．

リスト 3-4　得点の計算と表示 2(LIST 3-4.ASM)

```
;
;****** List 3-4-N ******
;
SCLOC    EQU     ØØ3EH                SCore LOCation …… スコア表示座標
kasan    macro   op1,kasanchi         得点計算および表示マクロ定義開始
         MOV     AL,kasanchi          AL ←加算値
         op1     AL,[BX]              AL ← AL op1 DS：[BX] …… op1 の演算を行う
         AAA                          AL の値を ASCII 加算補正
         PUSHF                        フラグをスタックへの退避
         MOV     [BX],AL              AL の値の保存
         PUSH    CX                   CX レジスタ値をスタックへ退避
         PUSH    BX                   BX レジスタ値をスタックへ退避
         CALL    DISPLE               AL で示される文字を(CL, CH)へ表示
         POP     BX                   BX レジスタ値をスタックから復元
         POP     CX                   CX レジスタ値をスタックから復元
         SUB     CL,2                 表示 X 座標を更新(－2)する
         DEC     BX                   データ・アドレスを更新(＋1)する
         POPF                         フラグをスタックから復元
         endm                         マクロ定義の終了
;
TKETA    equ     6                    得点の桁数
```

```
DISPSC: ;DISPlay SCore                          ── スコアの表示
        MOV     CX,offset SCLOC+2*TKETA         CX ←1の位のスコア表示座標
        MOV     BX,offset SCOREL                BX ←1,10の位の値がはいっている
        kasan   ADD,DL                          演算 ADD,加算値 DL でマクロ展開
        kasan   ADC,DH                          演算 ADC,加算値 DL でマクロ展開
        XCHG    AX,CX                           AX ←→ CX
        MOV     CX,TKETA-2                      CX ←点数の桁数分の繰り返し数－2
SCOREP: ;SCORE Print
        XCHG    AX,CX                           AX ←→ CX
        PUSH    AX                              AX レジスタ値をスタックへ退避
        kasan   ADC,Ø                           演算 ADC,加算値 DL でマクロ展開
        POP     AX                              AX レジスタ値をスタックから復元
        XCHG    AX,CX                           AX ←→ CX
        LOOP    SCOREP                          CX 回ループ
        RET                                     リターン
;
SCORE4  db      Ø                               得点表示ワークエリア 4
SCORE3  db      Ø                               得点表示ワークエリア 3
SCORE2  db      Ø                               得点表示ワークエリア 2
SCORE1  db      Ø                               得点表示ワークエリア 1
SCOREØ  db      Ø                               得点表示ワークエリア 0
SCOREL  db      Ø                               得点表示ワークエリア Low
DUMMY1  db      'ØØ',Ø                          ダミー「00」のデータ

        include LIST3-3.ASM                     LIST3-3.ASM を取り込む
```


テスト 3-4 テスト・プログラム(TEST 2-4.ASM)

```
;
;****** List 3-4-T ******
;
CODE    segment                                 命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,ES:PTNSEG,SS:STSEG

print   macro   string                          文字列出力マクロの定義
        LEA     DX,string                       マクロパラメータのオフセットを DX へ
        MOV     AH,9                            ファンクションコール番号 9……文字列出力
        INT     21H                             ファンクションコール
        endm                                    マクロ定義終了
;
PTEST:  ;Proglam TEST
        CLD                                     ディレクション・フラグ・リセット
        MOV     AX,CS                           AX ← CS
        MOV     DS,AX                           AX レジスタを介して DS に CS を格納
```

```
        print   CLEAR                          テキスト画面クリア
        CALL    DATLD                          パターン・データ・ロード
        CALL    GINIT                          グラフィック・システム初期化
        XOR     AX,AX                          AX←0
        MOV     word ptr CS:[SCOREØ],AX        スコアの初期化
        MOV     word ptr CS:[SCORE2],AX        スコアの初期化
        MOV     word ptr CS:[SCORE4],AX        スコアの初期化
        CALL    CLS                            グラフィック画面クリア
        MOV     CX,ØØ4AH                       点数表示座標セット：(CL,CH)に表示される
        MOV     BX,offset DUMMY1               ダミースコア用
        CALL    MSGPRN                         ダミースコア表示
TLOOP:  ;TEST
        MOV     DX,1                           加算スコア
        CALL    DISPSC                         スコアにDXを加算して(CL,CH)より表示
        MOV     AX,Ø4ØØH                       キーコード・グループ番号0入力用
        INT     18H                            キーデータ入力
        ROR     AH,1                           ESCデータをキャリーへ
        JNB     TLOOP                          ESCが押されていなければTLOOPへ
        print   CSRON                          カーソル・オン
        MOV     AX,ØCØØH                     }
        INT     21H                          } キーボード・バッファクリア
        MOV     AL,Ø                           リターン・コード←ノーマル
        MOV     AH,Ø4CH                        プロセスの終了
        INT     21H                            ファンクションコール
;
lodinf  struc                                  ロード情報構造体定義
CMD     db      Ø                              ロード・コマンド
PASS    db      "           "                  パス格納領域（11文字分）
ENDSIN  db      Ø                              パス・エンド・コード
LDADR   dw      Ø                              ロード・アドレス
LDSEG   dw      Ø                              ロード・セグメント
lodinf  ends                                   構造体定義終了
;
LODTBL  lodinf <1,"PTNDAT1.DAT" ,Ø,Ø,PTNSEG>
        lodinf <1,"MOJI.DAT"    ,Ø,2ØØØH,PTNSEG >
        lodinf < >
;
        include LIST3-4.ASM                    LIST3-4.ASMを取り込む
```

5. 衝突の処理…ゲームらしさの追求

衝突の判定，得点の表示が可能になれば，最後の仕上げとして全体をまとめなければなりません．これは，内容はともかく1つのゲームを完成させることにほかならず，商品を作るのと同じくたいへんなことなのです．商品にするには，まず，これに色を付けなければならないでしょう．色とは，もちろんカラーのことではなく，デコレーション・ケーキのように飾りを付けるということです．具体的にはタイトルとか，デモ画面とか，画面パターンの変化などを付けることで，これらは後からいくらでも追加できます．そういう意味で，この最後のテスト・プログラムはゲームの骨組みに当たるものであり，サブルーチンの内容だけに惑わされず，データの初期設定の方法とか，その順序とか，ゲーム全体の流れを的確に把握することが，ここでの大切なポイントです．プログラムを見る前に，まず全体をどのように構成するのか，フローチャートを見ながらその流れを追いかけることにしましょう．本来，プログラムというものはフローチャートを書いてから組んでいくと，バグの少ないものができるのですが，めんどうなため，どうしても直接プログラミングしてしまうことが多くなります．複雑なプログラムは，後で見ると作った本人でも何をやっているのかわからなくなってしまうものです．大作を作るときには，できるだけフローチャートを残す習慣を身につけることをお勧めします．

さて，図3-2のフローチャートから，敵と弾が1ループにつき2回移動するのに対し，主人公は1回しか移動していないことがわかります．主人公の移動速度が，敵や弾のスピードの半分であるということを意味します．また，主人公と敵との衝突判定は1ループについて2度行われていますが，厳密にはこれは100%の判定がされているとは言えません．それは，敵が動いて主人公と衝突の状態になった直後に，主人公が移動して敵と離れるというケースがあるからです．これを避けるには，主人公が移動する前に，もう一度衝突の判定をする必要があります．ただし，その程度のことは大目に見ようということで，今回はそこまでのきびしさは追求せず，このフローチャートどおりにプログラムを組んであります．このような一見すると気がつかない細かいことでも，フローチャートを追うことにより簡単にわかることが，めんどうなフローチャート作成の裏側にあるスバラシサの1つなのです．

プログラム本体については，条件アセンブルの部分が敵の移動ルーチンのなかで敵との衝突チェック，スコアのアップ，爆発時の処理(敵が弾に当たった場合)への分岐処理となっています．そして，このなかでまだプログラミングされていない敵の爆発ルーチンと，次に出現する敵を出すルーチンが，ここで新たに組まれています．

プログラムの内容については，コメントを読んでいけば理解できると思いますが，新しい敵の出現位置や移動コース(これま

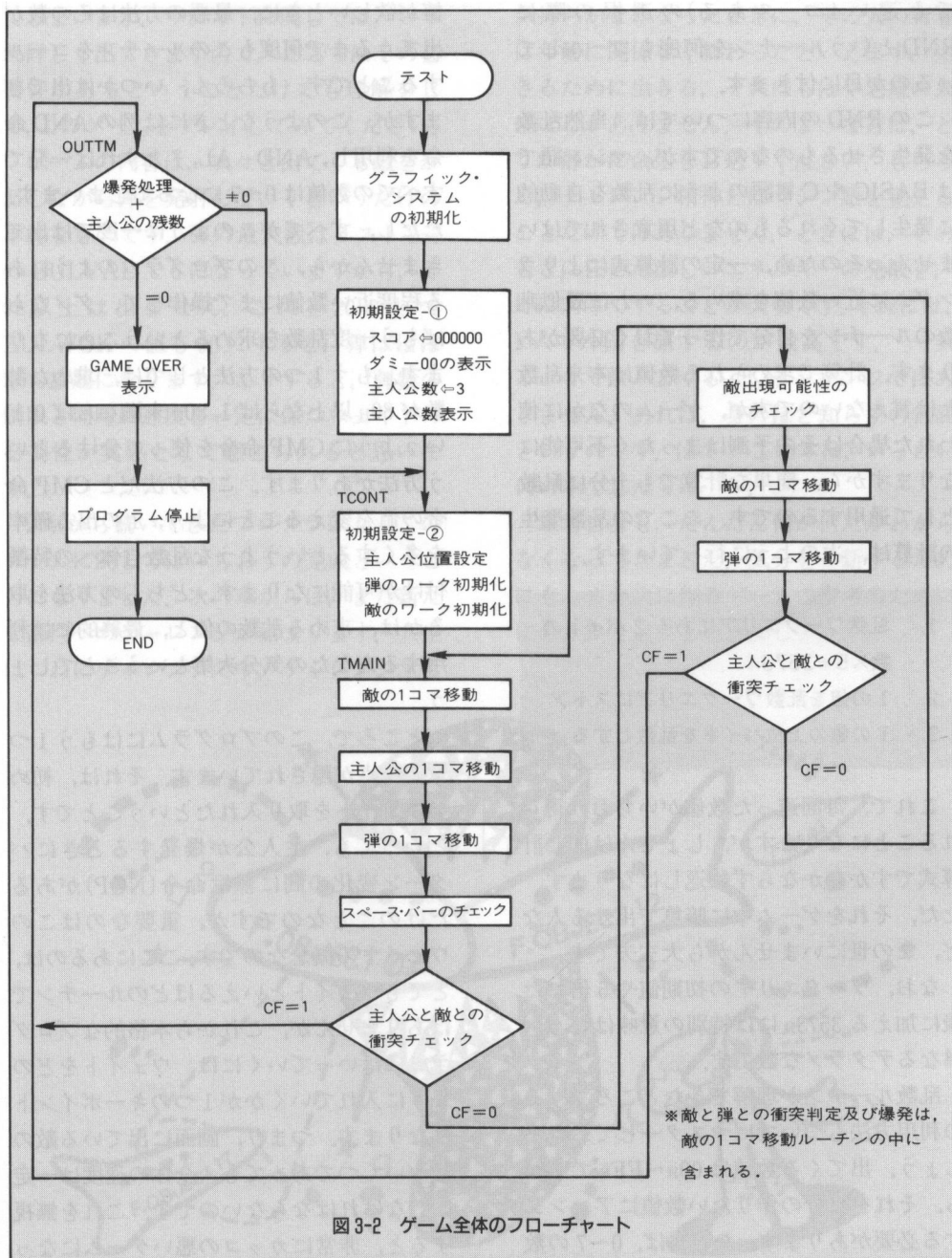



図3-2　ゲーム全体のフローチャート

でと違い4コースある)の選択の際にRNDというルーチンを何度もコールしているのが目に付きます．

このRNDの内容については，当然乱数を発生させるものなのですが，マシン語ではBASICやC言語のように乱数を自動的に発生してくれるものなど用意されてはいません．そのため，一定の計算式によりランダムに近い数値を求める，いわば疑似乱数のルーチンを自分で作っておく必要があります．計算で求められる数値は本来乱数とは言えないのですが，ゲームのなかに使われた場合はその予測はまったく不可能になりますから，簡単な計算でも十分に乱数として通用するのです．ここでの乱数発生の計算は，次のように行っています．

1. 乱数ワークエリアにある2バイトの数×5+3573H
2. 1の値を乱数ワークエリアにストア
3. 1の値の上位バイトを乱数とする

これで，毎回違った数値がいちおう得られることになりますが，しょせんは同じ計算式ですからかならず繰返しになります．ただ，それをゲーム中に暗算で出せる人など，この世にいませんから大丈夫です．

なお，ワークエリアの初期値や5倍した後に加える3573Hには特別の意味はなく，単なるデタラメな数です．

乱数ルーチンが理解できたところで，その利用方法もついでにマスターしておきましょう．出てくる数値は00H~FFHですから，それを自分の作りたい数値にアレンジする必要があります．たとえば，0~7の数値が欲しいときに，最悪の方法はその数が出てくるまで何度もこのルーチンをコールすることです．もちろん，いつかは出てきますが，このようなときには例のAND命令を利用し，AND　AL, 7とすれば一発ですべての数値は0~7になってしまいます．ただし，すべてがこのように一度では出てきませんから，このプログラムのようにある程度近い数値にまで操作して，ダメならばもう一度乱数を求めるということになります．もう1つの方法としては，求めた乱数が80H以上ならば1, 80H未満ならば0というようにCMP命令を使って分けるという方法があります．この方法だとCMP命令の値を変えることにより，1の出る確率を多くするというような乱数自体への特徴付けが可能になります．どちらの方法を取るかは，求める乱数の値と，最終的には利用するあなたの気分次第ということでしょう．

ところで，このプログラムにはもう1つ重要な点が隠されています．それは，初めて**ウェイト**を取り入れたということです．といっても，主人公が爆発するときにパターン変化の間に無駄命令(NOP)があるだけのことなのですが，重要なのはこのウェイトの概念なのです．ここにあるのは，とてもウェイトといえるほどのルーチンではありませんが，これから本格的なプログラムにはいっていくには，ウェイトをどのように入れていくかが1つのキーポイントになります．つまり，画面に出ている敵の数がいくつであっても，全体の速度は一定にしなければならないのです．これを無視すると，非常にカッコの悪いゲームになっ

てしまいます．ウェイトに関しては，いずれ詳しく出てきますので，ここではその必要性があるということだけでも理解をして，このゲームをプレイしてみてください．

ゲームの内容は，次々と出てくる敵をかわしながら弾を発射していくというだけの単純なものです．弾の最大数はリスト2-7のBULVALで6になっていますから，それ以上は出ません．これを20にしてBULWOKの値も60にすると，弾はほぼ撃ち放題になります．こうすると，弾数の制限と同時に速度を一定に保つウェイトの必要性が実感として感じられると思います．

2章から続いてきたプログラムも，このゲームに関してはいちおうの完成ということになりました．次の章からは，また新たな部門へのチャレンジが始まります．人生はつねにチャレンジです．チャレンジする気持ちがなくなったときに，その人の青春は年齢に関係なく終わったといえます．生きるために生きる，それはもはや老後の人生でしかありません．そのような方は，どうか静かに余生をお送りください．もちろん，チャレンジ精神とはマシン語を覚えることだけではありません．ときには，その若さに任せた激しいエネルギーの発散を，外に向けてすることが大切です．それには，1人で外国を旅するのが最高です．「一気イッキで飲みまくる」は，チャレンジではありません．あれは，身のほど知らずの無謀というのです．はて？この本は，いったい何の本でしたっけ……．

筆者自身も，だんだん何が何だかわからなくなってきました．気分を一新するためにも，また次に作るゲームの参考のためにも，ひと遊びといきましょう．

リスト3-5 シューティング・ゲームの仕上げ(LIST 3-5.ASM)

```
;
;****** List 3-5-N ******
;
DELAY:  ;DELAY                         ── ウェイト
        PUSH    CX                     CXレジスタ値をスタックへ退避
DELAYT: ;DELAY Times                   ── タイミング
        MOV     CX,8ØØH                ループ回数セット
DELAYL: ;DELAY Loop
        LOOP    DELAYL                 CX回ループする
        DEC     AL                     AL←AL−1
        JNE     DELAYT                 AL=0でなければDELAYTへジャンプ
        POP     CX                     スタックからCXレジスタ値を復元
        RET                            リターン
;
MYCRAS: ;MY CRAsh                      ── 主人公の爆発
        MOV     CH,16                  爆発の回数
        MOV     AL,EXPLO1              爆発の先頭パターン番号セット
MYCRA1: ;MY CRAsh 1
        PUSH    AX                     AXの値をスタックへ退避
        PUSH    CX                     AXの値をスタックへ退避
        MOV     CX,word ptr MYLOC      CX←主人公の座標
        CALL    DISP                   パターンの表示
        MOV     AL,4Ø                  ウェイト回数セット
        CALL    DELAY                  ウェイト
        POP     CX                     スタックからCXレジスタ値を復元
        POP     AX                     スタックからAXレジスタ値を復元
        XOR     AL,1                   AL←AL XOR 1
        DEC     CH                     CH←CH−1
        JNE     MYCRA1                 CH=0でなければMYCRA1へ
        CALL    CLS                    画面クリア
        MOV     DX,Ø                   DX←0
        CALL    DISPSC                 スコアの表示
;
        MOV     CX,ØØ4AH               ダミーの座標セット
        MOV     BX,offset DUMMY1       BX←DUMMY1のオフセット
        CALL    MSGPRN                 ダミー・スコアの表示
        MOV     BX,offset MYREST       BX←MYRESTのオフセット
        DEC     byte ptr [BX]          残り数更新
        MOV     AL,[BX]                AL←DS:[BX]
        PUSHF                          フラグをスタックへ保存
        PUSH    AX                     AXレジスタ値をスタックへ保存
        MOV     CX,MRLOC               CX←主人公の残数のワークエリア
        CALL    DISPLE                 (CL,CH)よりALを表示……残り数の表示
        XOR     AL,AL                  ウェイト数(=256)セット
        CALL    DELAY                  ウエイト
        POP     AX                     スタックからAXレジスタ値を復元
        POPF                           スタックからフラグを復元
        RET                            リターン
```

```
;
EMDEAD: ;EneMy DEAD
        MOV     AL,[SI].PATTERN                 AL←パターン番号
        INC     byte ptr [SI].PATTERN           パターン番号更新
        MOV     CX,word ptr [SI].XZAHYOU        CX←(X,Y)座標を一度にセット
        CMP     AL,EXPLO1+2                     パターン・エンドか？
        JE      NDISP                           エンドであればクリア・ルーチンへ
        JMP     DISP                            DISPへジャンプ
NDISP:  ;Not DISPlay
        MOV     byte ptr [SI].TSTATUS,Ø         敵出現フラグ・リセット
        MOV     BX,42ØH                         BX←消去のサイズ
        CALL    CLPTXY                          (CL,CH)よりサイズBXで消去
        RET                                     リターン
;
EMAPP:  ;EneMy APPeare
        MOV     BX,offset ENEMY                 敵のワークエリアの先頭アドレス
        MOV     CX,EMVAL                        CX←敵の総数
EMAPPL: ;EneMy APPeare Loop
        PUSH    CX                              CXレジスタ値をスタックへ退避
        MOV     AL,[BX].TSTATUS                 AL←敵の出現フラグ
        OR      AL,AL                           AL=0か？
        JNE     $+5                             AL=0でなければスキップ
        CALL    NEWEM                           敵の出現
        POP     CX                              CXレジスタ値をスタックから復元
        ADD     BX,type ENEMY                   BX←次の敵のワークエリアを求める
        LOOP    EMAPPL                          CX回ループ
        RET                                     リターン

RND:    ;RaNDom figure
        PUSH    BX                              BXの値をスタックへ退避
        MOV     BX,word ptr RNDWOK              BX←前回の乱数基数
        MOV     DX,BX                           DX←BX
        SHL     BX,1                            BX←BXの2倍
        SHL     BX,1                            BX←さらに2倍 …… もとの数の4倍になる
        ADD     BX,DX                           BX←BX+DX …… もとの数の5倍になる
        MOV     DX,3573H                        DX←3573H(適当な数値)
        ADD     BX,DX                           BX←BX+DX
        MOV     word ptr RNDWOK,BX              RNDWOK←BX
        MOV     AL,BH                           AL←BH
        POP     BX                              スタックからBXレジスタ値を復元
        RET                                     リターン

REND    equ     46                              右端値
LEND    equ     Ø                               左端値
UEND    equ     Ø                               上端値
DEND    equ     46                              下端値
```

```
;
NEWEM:  ;NEW EneMy
        MOV     byte ptr [BX].TSTATUS,1     敵出現フラグ・セット
        CALL    RND                         ALに乱数を求める
        AND     AL,3                        4以上カット
        JE      NEWEM                       AL=0か？
        MOV     [BX].PATTERN,AL             敵のパターン番号セット
        MOV     AH,Ø                        AH←0
        MOV     [BX].TOKUTEN,AX             得点をセット
        CALL    RND                         乱数をALへ求める
        AND     AL,3                        4以上カット
        ADD     AL,AL                       ALを2倍
        PUSH    DI                          DIレジスタ値をスタックへ退避
        MOV     DI,offset COUADR            DI←移動方向テーブル・アドレス
        ADD     DI,AX                       DI←DI+AX
        MOV     CX,[DI]                     CX←移動方向
        POP     DI                          スタックからDIレジスタ値を復元
        MOV     [BX].POINTER,CX             ポインタ値をセット
NEWEMY: ;NEW EneMy Y
        CALL    RND                         ALに乱数を求める
        AND     AL,7FH                      80H以上カット
        INC     AL                          1～80Hに補正
        CMP     AL,DEND-2Ø                  AL≧下エンド-20か？
        JNB     NEWEMY                      条件が成り立てばNEWMYへ
        MOV     [BX].YZAHYOU,AL             Y座標セット
NEWEMX: ;NEW EneMy X
        CALL    RND                         ALに乱数を求める
        AND     AL,7FH                      80H以上カット
        INC     AL                          1～80Hに補正
        CMP     AL,REND-1                   AL≧右エンド-1か？
        JNB     NEWEMX                      条件が成り立てばNEWMXへ
        MOV     [BX].XZAHYOU,AL             X座標セット
        RET                                 リターン
RNDWOK  db      113,31                      RaNDom figure WOrk area

        include LIST3-4.ASM                 LIST3-4.ASMの取り込み
```

テスト3-5　テスト・プログラム(TEST 3-5.ASM)

```
;****** List 3-5-T ******

CODE    segment                                      命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,ES:PTNSEG,SS:STSEG

print   macro   string                               文字列出力マクロの定義
        LEA     DX,string                            マクロパラメータのオフセットをDXへ
        MOV     AH,9                                 ファンクションコール番号9…… 文字列出力
        INT     21H                                  ファンクションコール
        endm                                         マクロ定義終了
;
PTEST:  ;Program TEST
        CLD                                          ディレクション・フラグのリセット
        MOV     AX,CS                                AX←CS
        MOV     DS,AX                                AXレジスタを介してDSにCSを格納
        print   CLEAR                                テキスト画面クリア
        CALL    DATLD                                パターン・データ・ロード
        CALL    GINIT                                グラフィック・システム初期化
        CALL    CLS                                  グラフィック画面クリア
        MOV     DI,offset SCORE2                     DI←SCORE2のオフセット・アドレス
        MOV     AX,CS                                AX←CS
        MOV     ES,AX                                AXレジスタを介してESへCSをセット
        MOV     CX,3                                 ワークエリア分のループ数をセット
        XOR     AX,AX                                AX←0
        REP     STOSW                                ES:[DI],AX,DI←DI+2,でCX回ループ
        MOV     CX,ØØ4AH                             ダミー表示座標セット
        MOV     BX,offset DUMMY1                     BX←ダミーのアドレス
        CALL    MSGPRN                               ダミー「00」の表示
        MOV     DX,Ø                                 DX←0
        CALL    DISPSC                               スコア「000000」の表示
;
        MOV     AL,3                               } 主人公の総数セット
        MOV     MYREST,AL                          }
        MOV     CX,MRLOC                             CX←DS:MRLOC
        CALL    DISPLE                               残数の表示
TCONT:  ;Test CONTinue
        MOV     BX,word ptr INITML                   主人公の初期座標を取り出す
        MOV     word ptr MYLOC,BX                    主人公の初期座標をワークエリアへセット
;
        MOV     BX,offset BULWOK                     BX←弾のワークエリアの先頭アドレス
        MOV     CX,BULVAL                            CX←弾の総数
TL2:    ;Test Loop 2
        MOV     byte ptr [BX],Ø                      DS:[BX]←0
        ADD     BX,3                                 BX←BX+3:次の弾のワークエリアを求める
        LOOP    TL2                                  CX回ループ
```

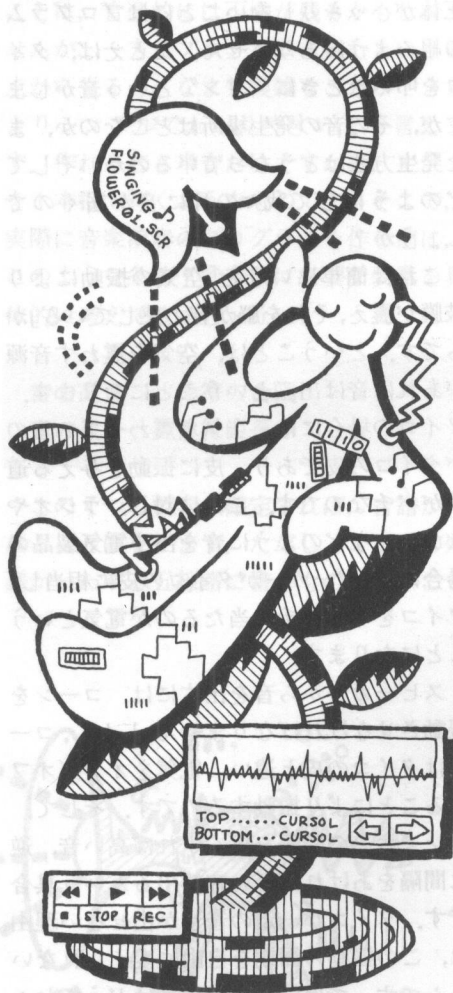
```
;
        MOV     BX,offset ENEMY             BX ←敵のワークエリアの先頭アドレス
        MOV     CX,EMVAL                    CX ←敵の総数
TL3:    ;Test Loop 3
        MOV     byte ptr [BX],Ø             DS：[BX]←0：敵の出現フラグ・リセット
        ADD     BX,type ENEMY               BX ←次の敵のワークエリアを求める
        LOOP    TL3                         CX 回ループ
;
TMAIN:  ;Test MAIN loop
        MOV     AX,ØCØØH                  } キーボード・バッファクリア
        INT     21H                       }
        MOV     CX,Ø4ØØØH                   ウエイト用ループ回数セット
WAITØ:  ;WAIT Ø
        PUSH    AX                          ウエイト用無駄命令
        POP     AX                          ウエイト用無駄命令
        LOOP    WAITØ                       CX 回ループ
        CALL    EMMVAL                      敵を移動
        CALL    MYMOVE                      主人公を移動
        CALL    ABMOVE                      弾を移動
        CALL    SSKCK                       SPACE のチェック
        CALL    MYCHK                       主人公と敵の衝突チェック
        JB      OUTTM                       衝突していれば OUTTM へ
        CALL    EMAPP                       敵出現のチェック
        CALL    EMMVAL                      敵を移動
        CALL    ABMOVE                      弾を移動
        CALL    MYCHK                       主人公と敵の衝突チェック
        JNB     TMAIN                       衝突していなければ TMAIN へ
OUTTM:  ;OUT of Test Main
        CALL    MYCRAS                      主人公の爆発
        JE      NTCONT                      残り数が 0 であれば NTCONT へ
        JMP     TCONT                       TCONT へジャンプ
NTCONT: ;Not Test CONTinue
        MOV     BX,offset GOVER             BX ← GOVER のオフセット値
        MOV     CX,1Ø1ØH                    表示座標セット
        CALL    MSGPRN                      座標(CL, CH)へ「ゲームオーバー」を表示
        print   CSRON                       カーソル・オン
        MOV     AX,ØCØØH                  } キーボード・バッファクリア
        INT     21H                       }
        MOV     AL,Ø                        リターン・コード←ノーマル
        MOV     AH,Ø4CH                     プロセスの終了
        INT     21H                         ファンクションコール

        ;include LIST2-7.ASM                LIST2-7.ASM を取り込む
;
GOVER   db      'G A M E  '                 Game OVER
        db      'O V E R',Ø
;
INITML  db      26,46                       INITial My Location …… 主人公の初期座標
;
MYREST  db      Ø                           MY REST …… 主人公の残り数がはいるワークエリア
```

```
;
EXPLO1   equ      4                       EXPLOsion 1 …… 爆発の先頭パターン
MRLOC    equ      1Ø4AH                   My Rest LOCation …… 残り数の表示座標
;
QRR      equ      1                       方向別のラベル化
QUR      equ      2                         〃
QUU      equ      3                         〃
QUL      equ      4                         〃
QLL      equ      5                         〃
QDL      equ      6                         〃
QDD      equ      7                         〃
QDR      equ      8                         〃
NM       equ      9                         〃
NP       equ      1Ø                        〃
;
;
COURS1: ;COURSe 1                         ── 移動方向データ1
         db       QDD,QDD,QDR,QDR
         db       QDR,QRR,QRR,QRR
         db       QRR,QUR,QDD,QDD
         db       QDL,QDL,QDL,QLL
         db       QLL,QLL,QLL,QUL
         db       NP
         dw       offset COURS1
COURS2: ;COURSe 2                         ── 移動方向データ2
         db       QDD,QDD,QDD,QDD
         db       QDL,QLL,QLL,QLL
         db       QLL,NM,NM,NM
         db       NM,NM,QDL;QDD
         db       QDD,QDD,QDD,QDR
         db       QRR,QRR,QRR,QRR
         db       NM,NM,NM,NM
         db       NM,QDR
         db       NP
         dw       offset COURS2
COURS3: ;COURSe 3                         ── 移動方向データ3
         db       QRR,QDD,QLL,QDD
         db       QRR,QRR,QDD,QLL
         db       QLL,QDD,QRR,QRR
         db       QRR,QRR,QDD,QLL
         db       QLL,QLL,QLL,QDD
         db       QRR,QRR,QRR,QRR
         db       QRR,QRR,QRR,QRR
         db       QDD,QLL,QLL,QLL
         db       QLL,QLL,QLL,QLL
         db       QLL,QDD,QRR,QRR
         db       QRR,QRR,QRR,QRR
         db       QRR,QRR,QRR,QRR
         db       QRR,QRR,QRR,QRR
         db       QRR,QRR,QDD,QLL
         db       QLL,QLL,QLL,QLL
```

```
        db      QLL,QLL,QLL,QLL
        db      QLL,QLL,QLL,QLL
        db      QLL,QLL,QLL,QDD
        db      QRR,QRR,QRR,QRR
        db      QRR,QRR,QRR,QRR
        db      QRR,QRR,QRR,QRR
        db      QRR,QRR,QRR,QRR
        db      QRR,QRR,QRR,QRR
        db      QRR,QRR,QRR,QRR
        db      QRR,QRR,QRR,QRR
        db      QRR,QRR,QRR,QRR
        db      QDD,QLL,QLL,QLL
        db      QLL,QLL,QLL,QLL
        db      QLL,QLL,QLL,QLL
        db      QLL,QLL,QLL,QLL
        db      QLL,QLL,QLL,QLL
        db      QLL,QLL,QLL,QLL
        db      QLL,QLL,QLL,QLL
        db      QLL,QLL,QLL,QLL
        db      QLL,QDD
        db      NP
        dw      offset COURS3
COURS4: ;COURSe 4                                         ── 移動方向データ4
        db      QDR,QDR,QDR,QDL
        db      QDL,QDL
        db      NP
        dw      offset COURS4
;
COUADR: ;COUrse ADdRess                                   ── 移動方向のアドレス・テーブル
        dw      offset COURS1,offset COURS2
        dw      offset COURS3,offset COURS4
;
lodinf  struc                                             ロード情報構造体定義
CMD     db      Ø                                         ロード・コマンド
PASS    db      "           "                             パス格納領域(11文字分)
ENDSIN  db      Ø                                         パス・エンド・コード
LDADR   dw      Ø                                         ロード・アドレス
LDSEG   dw      Ø                                         ロード・セグメント
lodinf  ends                                              構造体定義終了
;
LODTBL  lodinf <1,"PTNDAT1.DAT"  ,Ø,Ø,PTNSEG>
        lodinf <1,"MOJI.DAT"     ,Ø,2ØØØH,PTNSEG >
        lodinf < >
;
        include LIST3-5.ASM                               LIST3-5.ASMの取り込み
```

4章 ●音楽演奏と効果音

●音楽は全人類共通の言葉であると言われていますが，確かにこの世から音楽が消えてしまったとしたら，寂しい世界になってしまうでしょうネ．スポーツでも野球やプロレスなどかなりの分野で，音楽によって雰囲気を盛り上げて，観客をより楽しませてくれようとしています．アメリカン・フットボールなどでは，主役がいったいどっちなのかわからなくなるほどハデにやっています．ゲームだって同じことです．もし，ゲームセンターの音をすべて消してしまったら，どんな激しいゲームをしても，興奮の度合は半分以下になってしまうことでしょう．

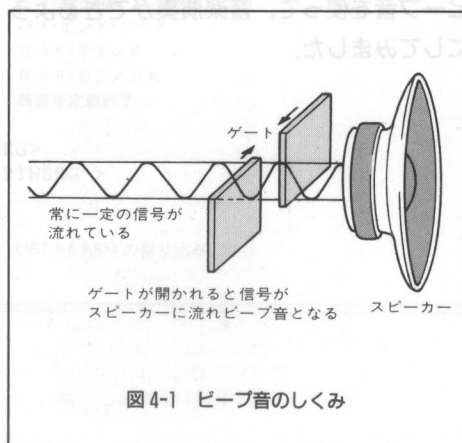
●そこで本書ではFM音源を取り上げるのは言うに及ばず，PC-9801シリーズのビープ音を使って，音楽演奏ができるようにしてみました．

1. BEEP音…音の仕組みとハードウェア

音とは，いったいどういう経路で我々の耳に聞こえてくるのでしょうか．音楽を出すプログラムを作るといっても，まず音の正体がハッキリしないことにはプログラムの組みようがありません．たとえば，タイコを叩いたときに〝ドン〟という音がしますが，その音の発生場所はどこなのか，また発生方法はどうなっているのか，そしてどのようにして我々の耳に音が届くのでしょうか？

これは簡単にいうと「空気の振動により鼓膜が震え，それを脳が音と感じている」からです．ということは，空気を震わす音源があれば音は出るということになります．タイコの場合には，空気を震わせているのがタイコの皮であり，皮に振動を与える道具がバチなのです．これに対し，ラジオやステレオなどのように音を出す電気製品の場合は，スピーカーがタイコの皮に相当し，タイコを叩くバチに当たるのが電気ということになります．

スピーカーから音を出すには，コーンを振動させなければなりません．しかし，コーンはタイコの皮と違い，電気をオン／オフすることにより振動するのです．そして，オン／オフの間隔を狭くすれば高い音，逆に間隔をあければ低い音が出るという具合です．タイコが一定の音しか出せない理由は，この振動の周期を自由に変えられないからです．では，音の大きさはどうでしょうか．タイコは強く叩けば大きい音が出ます．スピーカーも同じように強い電気，すなわち電流を大きくすれば大きな音を出せるわけです．これを手で調節できるようにしたものがボリュームつまみなのです．

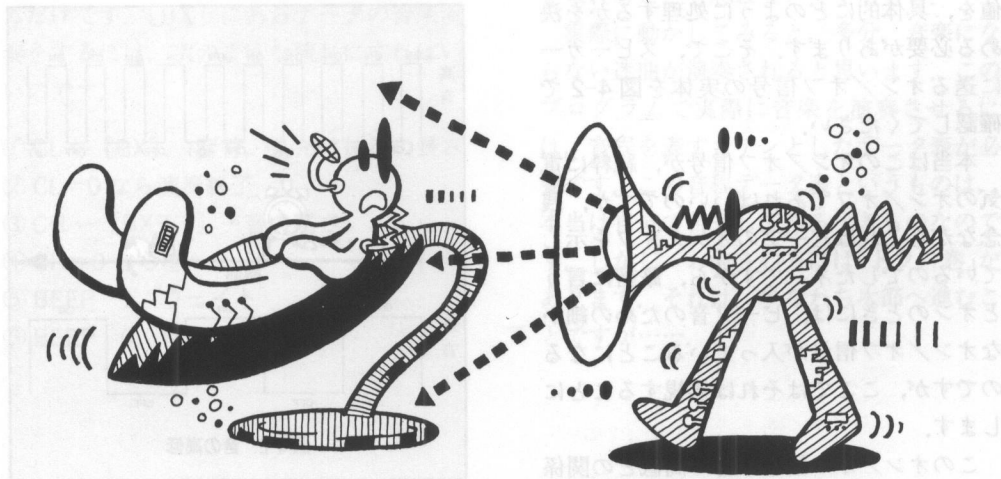
PC-9801のビープ音も，電気で作られた音ですから最終的には小さなスピーカーを振動させて鳴っています．このスピーカーに電気信号を送れば音が出ることになるのですが，問題はPC-9801ではスピーカーに対する電圧の変更は，外部ボリュームでしかできないという点です．我々がプログラムでできることは，ただ1つビープ音を出したり止めたりすることだけです．では，なぜビープ音だけは鳴るかというと，すでに一定の電圧で一定のオン／オフの周期を持った電流がスピーカーの直前まできていて，ゲートと呼ばれるものの開閉によってそれがスピーカー側に流れたり，止まったりするようになっているからなのです．

図4-1　ビープ音のしくみ

とにかく，スピーカーを制御するにはこのゲートを操作するしか方法はないわけですから，ここを強引にオン／オフさせて音楽を作らなければなりません．その場合，当然のことながら不要なビープ音が混じってきますので，純粋に電気のオン／オフで作られた音ではなく，濁った音となってしまいます．しかし，濁った音といってもこの音しか知らなければ，濁りもまったく気にならない程度のものですし，ゲームには大いに役立てることができるのです．

このビープ音の制御をするには，出力ポート37Hに，6または7を出力することによって操作します．出力ポート37Hに6を出力すれば〝BEEP 1〞，7を出力すれば〝BEEP 0〞ということです．これは，ビープ音だけで作る音楽ですから，ビープ音楽と呼ぶことにしましょう．ビープ音楽は，音色も音量も変えられない，いわば音楽の原点です？ PC-9801用のFM音源ボードを持っている方でも，音の基礎であるビープ音楽を理解することは，FM音源やPSGのコントロールに大いに役立ちますので，ひととおり流し読みしてください．

PC-9801のハードウェア上の制約から，我々が作り出せる音は，高さと長さだけが自由で，音色はもちろんのこと音の大きさも変えられないことがわかりました．このことは，音楽的には不満が残るかもしれませんが，一方でプログラムを組むという観点から見ると，なまじ複雑なことができるよりシンプルでわかりやすいとも言えます．作れる音も単音だけですから，ピアノを1本指で弾くようなものです．そこで，実際に音楽演奏のプログラムを作る前に，このような条件下で音楽に必要な要素を具体的に考えてみましょう．

音の高さ	休符の合図
音の長さ	休符の長さ

この4つの要素が確定すれば，音譜が作れることになります．ただし，ここでの楽譜がいわゆる五線譜に書くものでないこと

は，すでに想像がついていると思います．もちろん，最初に作曲するときは五線譜に書いてかまわないのですが，コンピュータに演奏させるには，何らかの方法で16進数のデータにしなければなりません．そして，データにするということは，その音データの開始番地とデータ終了の合図を示す必要があるということです．これらを含んだすべての要素を，実際にどのような形でプログラムに組み入れればいいのか，色々な方法が考えられますが，ここでは次のように決めています．

1. 音楽演奏の手順
   (1)BX ←音楽演奏用データの開始アドレス
   (2)データに基づいて音楽演奏をするルーチンをコールする
2. データの意味(終了の合図以外は2バイトで1組とする)
   前半の1バイト：01〜FF$_H$＝音符または休符の長さ
   　　　　　　　　00$_H$＝演奏の終了の合図
   後半の1バイト：01〜FF$_H$＝音の高さ
   　　　　　　　　00$_H$＝休符の合図

かなり具体的になりましたが，これだけの決め方ではまだ実際にプログラムを組むことはできません．それは，16進数で表されたデータと，音との関係がハッキリしていないからです．プログラミングするには，この音の長さを表す数値と，高さを表す数値を，具体的にどのように処理するかを決める必要があります．そこで，スピーカーに送るオン／オフ信号の実体を図4-2で確認してください．

本当はこのオン／オフ信号が，純粋に電気のオン／オフであればいいのですが，残念ながらこれはビープのオン／オフを示しているのでしたネ．ですから，厳密に言うとオンのときには，ビープ音のための細かなオン／オフ信号が入っていることになるのですが，ここではそれは無視することにします．

このオン／オフ信号と音の高低との関係は，オンからオフまたはオフからオンまでの間隔にあります．つまり，高い音ほどオン／オフの間隔が短く，逆に低い音ほどその間隔が長いということです．一方，音の長さというのは時間のことですから，オン

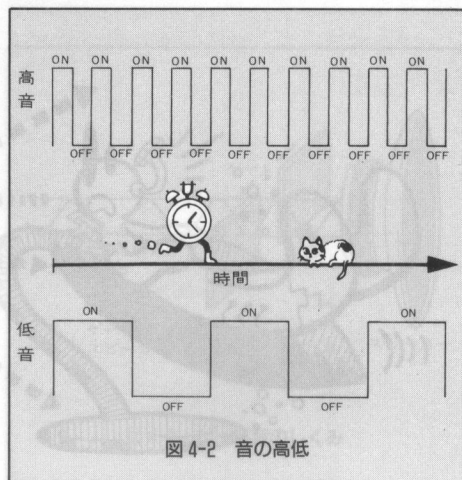



図4-2　音の高低

／オフを実行しているトータル時間を計ればいいのです．ところが，実際には正確な時間を簡単に計る方法がないので，繰り返したオン／オフの回数によって，その長さを表すことにしています．したがって，図4-2の例のように同じ長さの音でも，音の高さによって,長さを示す値(オン／オフの回数)は違ってくることになります．

また，音の高さ(オン／オフの間隔)の表現は，これも時間で表します．しかし，数えるものが何もありません．そこで，何か基準となる無駄命令を決めて，その実行回数を数えるという原始的な方法で数値化します．音を微妙に変えられるようにするには，この無駄命令もできるだけ簡単なもののほうがいいことになりますから，最も単純に「実行回数-1」をするだけにします．

これで，音の長さと高さを数値に変更する方法が決まりました．休符については，オン／オフの代わりにオフ／オフと実行させれば，無音状態にすることができます．データの内容が決まれば，残るはプログラムだけです．［BX］にあるデータの音楽演奏をするには，次のような流れにすればいいのです．

①CL←［BX］…音符，または休符の長さ
②CL＝0なら演奏終了
③CH←［BX＋1］…音の高さ
④CH＝0なら⑨へ
⑤BEEP　1：ウェイト
⑥BEEP　0：ウェイト
⑦CL←CL−1：CL≠0なら⑤へ
⑧BX←BX＋1：①へ
⑨休符(ウェイト×2×CL)：⑧へ
〝ウェイト〟…CL＝0になるまでCH＝0になるまでCH←CH−1を繰り返す

これを実際にプログラミングしたのが，リスト4-1です．一部，現時点では意味のわからない箇所(ラベル名でPOINT1とPOINT2)もあると思いますが，今は無視してください．なお，本章のプログラムは新たに作成するものです．2, 3章のプログラムはインクルードしません．グラフィック関係にはこれまでと共通して使えるものもあるのですが，プログラムミスを防ぐ意味からも，あえて書き直しています．

テストの実行は，例によってテスト・プログラムからです．まだ，演奏用のデータが何もはいっていませんから，適当な数字を2バイトずつ入れてデータとします．テストですから，10バイトほどあれば十分です.データの最後には,演奏終了の合図(00)を入れることも忘れないでください．

実際に動かしてみると，多分，音楽にならない迷曲が演奏されると思います．このプログラムで実際に音楽を演奏させるには，音程を表すキチンとしたデータ表が必要です．この音程データ表というものは，本当は自分で苦労して作るべきものなのです．しかし，お急ぎの方には「トラの巻」があります．それは，すなわち次節へ進むことですが……．

リスト 4-1　BEEP 音楽の演奏(LIST 4-1.ASM)

```
;
;**** List 4-1 ******
;
MUSIC:  ;MUSIC play
        MOV     CH,[BX]         CH←DS:[BX]…… 音の長さ
        AND     CH,CH           CH=0か?
        JNE     NOTRET          CH=0でなければNOTRET
        RET                     リターン
NOTRET: ;NOT RET
        INC     BX              BX←BX+1
        MOV     CL,[BX]         CL←DS:[BX]…… 音の高さ
        AND     CL,CL           CL=0か?
        JNE     BEEP1           CL=0でなければBEEP1
        JMP     PAUSE           PAUSEへジャンプ
;
BEEP1:  ;BEEP 1
        MOV     AL,6            AL←6
        OUT     37H,AL          ビープ・オン
        CALL    PWAIT           CLの値によりウェイトを置く
POINT1: ;POINT 1
        NOP                     効果音用スペース
        NOP                     〃
        MOV     AL,7            AL←7
        OUT     37H,AL          ビープ・オフ
        CALL    PWAIT           CLの値によりウェイトを置く
POINT2: ;POINT 1
        NOP                     効果音用無駄命令
        NOP                     〃
        DEC     CH              CH←CH-1
        JNE     BEEP1           CH≠0ならBEEP1へ
        JMP     NEXTDT          NEXTDTへジャンプ
;
PAUSE:  ;PAUSE                  (休符)
        CALL    PWAIT           CLの値によりウェイトを置く
        CALL    PWAIT           CLの値によりウェイトを置く
        DEC     CH              CH←CH-1
        JNE     PAUSE           CH≠0ならBEEP1へ
NEXTDT:
        INC     BX              BX←BX-1
        JMP     MUSIC           MUSICへジャンプ
;
PWAIT:  ;Program WAIT
        PUSH    CX              CXの値をスタックへ退避
WCOUNT: ;Wait COUNT
        PUSH    AX              AXの値をスタックへ退避
        PUSH    DX              DXの値をスタックへ退避
        MUL     AX              ウェイト用無駄命令
        MUL     AX              ウェイト用無駄命令
        POP     DX              DXの値をスタックから復元
        POP     AX              AXの値をスタックから復元
```

```
        DEC     CL
        JNE     WCOUNT
        POP     CX                              CXの値をスタックから復元
        RET                                     リターン

MDTOP   label byte                              Music Data TOP
        db      ØAØH,Ø41H,Ø9ØH,Ø4ØH,ØAØH,Ø41H,Ø9ØH,Ø4ØH
        db      ØØ5H,ØØØH,ØAØH,Ø4EH,Ø9ØH,Ø4FH,ØADH,Ø4EH

        db      Ø9ØH,Ø4FH,ØØ7H,ØØØH,Ø5ØH,Ø59H,ØØAH,ØØØH
        db      Ø5ØH,Ø65H,ØØAH,ØØØH,Ø5ØH,Ø59H,ØØAH,ØØØH

        db      Ø5ØH,Ø4EH,ØØAH,ØØØH,ØFØH,Ø65H,ØØAH,ØØØH
        db      ØAØH,Ø87H,ØØCH,ØØØH,ØBØH,Ø65H,ØØAH,ØØØH

        db      Ø58H,Ø59H,ØØAH,ØØØH,Ø6ØH,Ø4EH,ØØAH,ØØØH
        db      Ø63H,Ø49H,ØØAH,ØØØH,Ø68H,Ø41H,ØØAH,ØØØH

        db      Ø6BH,Ø39H,ØØAH,ØØØH,Ø71H,Ø33H,ØØAH,ØØØH
        db      ØFFH,Ø3ØH,ØFFH,Ø3ØH,ØFFH,Ø3ØH,ØØØH,ØØØH
```


テスト4-1　テスト・プログラム(TEST 4-1.ASM)

```
;
;****** TEST 4-1 ******
;
CODE    segment                                 命令の置かれているセグメントの始まり
        assume CS:CODE,DS:CODE,SS:STSEG
;
PTEST:  ;Program TEST
        MOV     AX,CS                           AX ← CS
        MOV     DS,AX                           AXレジスタを介してDSにCSを格納
        MOV     BX,offset MDTOP                 音楽データの先頭アドレス
        CALL    MUSIC                           音楽を演奏
        MOV     AL,Ø                            リターン・コード・ノーマル
        MOV     AH,Ø4CH                         プロセスの終了
        INT     21H                             ファンクションコール

        include LIST4-1.ASM                     LIST4-1.ASMを取り込む

CODE    ends                                    CODEと名付けたセグメントの終わり

STSEG   segment stack                           スタックセグメントの宣言
        db      1ØØH dup (?)                    メモリ領域を100Hバイト確保
STSEG   ends                                    スタックセグメントの終わり

        end                                     プログラム・エンド
```

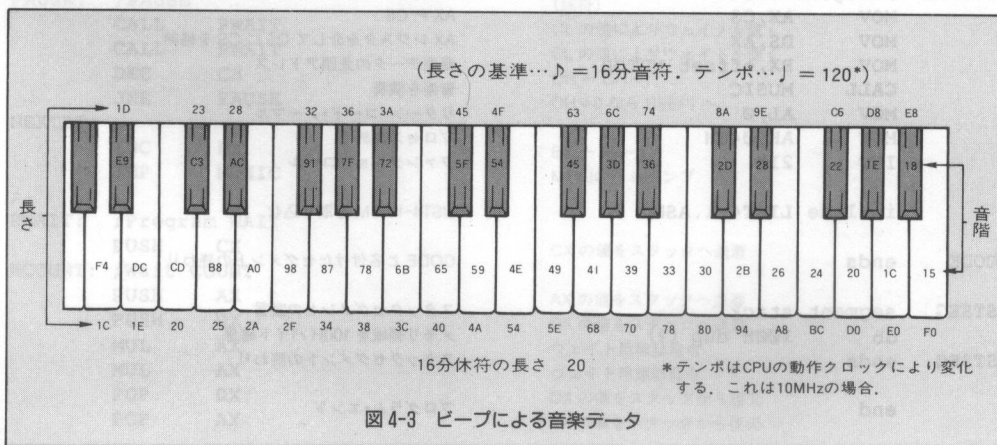
2. 音楽…BEEP 音楽用音程データ

音程とは，いったいどのようにして決められているのでしょうか．これが，難しいようで実は非常に簡単な取り決めしかしていないのです．基準となる音はハ長調のラで，周波数(1秒間のオン／オフの回数)は440Hzとなっています．そして，音程が1オクターブ上がれば周波数は倍に，下がれば半分になります．1オクターブをピアノで見ると，黒鍵も含めて12のキーが並んでいますから，周波数も12に分ければいいのですが，均等に分けるのではなく，1オクターブ上がったときに倍になるようにしなければいけません．

これだけの決まりなので，基準音だけわかれば音程データの作成は単なる作業になりそうですね．ところが，このビープ音楽というのは濁りはあるし，正確なメトロノームもないという，いわば問題だらけの音楽ですから，完全な音程など作りようがありません．いちおう，このことを頭に入れた上で，図4-3の音程データ表を見るようにしてください．

このデータ表によると，音の高さが1オクターブの差で数字がキチンと倍(半分)になっているときと，だいぶズレているときとがありますが，これもビープ音楽に問題のある証拠で，計算どおりのデータで実際にテストをしてみると，音程が狂ってしまうのです．

そのため，この音程データは試行錯誤の結果作成した貴重な資料なのです．しかし，私が音楽的にプロの耳をもっているというわけではありませんから，あるいは音程に若干の狂いがあるかもしれません．そのあたりのデータ修正に関しては，あなたなりの音に仕上げるようにしてください．この音程データを基にして作った，テスト音楽がありますので実験してみましょう．リス

図4-3 ビープによる音楽データ

ト 4-1 の MDTOP 以降のデータをセットしてみてください.

アレ!!　どこかで聞いたことがある……と，すべての方が思ってくださると非常にウレシイのですが…….まず無理でしょうね.

この音楽のデータを見ると，音程データ表(図 4-3)そのままの使い方ではありません.たとえば，長さなどはまったくデタラメのようですし，小さな休符も意味もなく多く使われているように見えます.実は，この辺がビープ音楽の長所でもあり，また. めんどうな点でもあるのです.できるだけ長所を生かすには，次の事を考えながら，少しずつ修正を加えて完成させるようにするしかありません.

1. **連符や音の歯切れ良くしたいときは，間に短い休符を入れる.**

   例：　ド・ド・ド＝ド・短い休符・ド・短い休符・ド

2. **音の長さは，実際に耳で聞き何度も修正をする.**

1. の例でも短い休符がはいる分だけ音が長くなることになりますし，楽譜どおりのデータはなかなかできないものです.数値ですから，音符にできないような微妙な長さでもかまわないのです.聞きながら，気に入るまで修正してください.

3. **楽譜では表現不可能な高さの音も作ることができる.**

この音楽の最初の部分でも使っているテクニックですが，音程データを少し(+1または−1)狂わすことにより，音を震わすことができます.また，ミとファの中間というような楽譜にない高さの音が作れるのもデジタル・ミュージックの面白さです.

これで，実際の演奏用データがキチンとならない理由がわかったと思います.自由ということは，すなわちめんどうということなのです.ピアノよりエレクトーン，エレクトーンよりシンセサイザー……音が自由になるにつれ操作するスイッチ類が多くなっていくようなものです.

3. 臨場感…BEEPによる効果音

ゲームの進行を側面から盛り上げるという意味では，ビープ音楽も1つの効果音ということができますが，一般的には音楽でない音のことを効果音といいます．つまり，ゲーム・センターなどにあふれている例の音のことです．

ビープ音の波形(図4-2)を，もう一度見てください．高音でも低音でも，オンの時間とオフの時間が同じになっています．別に，わかりやすくするために同じ間隔にしているのではありません．この間隔をを一定にしているからこそ，一定の高さの音がでているのです．もし，この間隔がバラバラだったら，それはもう雑音でしかないのです．

それでは，図4-4のようにある決まったルールの下で，この間隔を変化させたらどうなるのでしょうか．

実際にどんな音になるのか，想像が付かないかもしれませんが，例1では高音から低音へ，例2では低音から高音へ，いずれも急激に変化していくハズです．この図4-4では，ビープ・オフの後に間隔の変更がされています．ということは，リスト4-1のなかで間隔を示しているのはCLですから，ビープ・オフ後にCLの値を+1，または，−1すればいいということになります．

このCL値の変更場所を具体的に見るとリスト4-1のPOINT2がそれにあたります．つまり，ここで2つ並んでいる〝NOP〟(90_H)を〝INC　CL〟($0C1FE_H$)に書き換えると例1のようになり，〝DEC　CL〟($0C9FE_H$)とすれば例2のようになるわけです．同じようなことをPOINT1でも実行すれば，音の変化はより急激になることになります．結局，全体では次ページの4種類の音変化ができます．

この4種の音変化をさせるプログラムが**リスト4-2**ですが，それぞれ実行が終わると変更したポイントをNOP(90_H)に戻すようにしてあります．

このプログラムはリスト4-1をインクルードする形でセーブしておいてください．5章では，このセーブしたプログラムをインクルードしながら迷路ゲームを作ります．アセンブルしたらテスト・プログラムを実行してみてください．

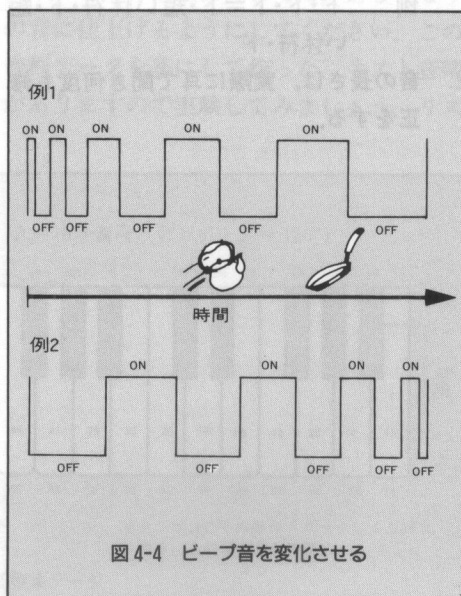



図4-4　ビープ音を変化させる

(1) POINT1のところを0C9FEH(DEC CL)　…… 音のアップ変化1
(2) POINT2のところを0C1FEH(INC CL)　…… 音のダウン変化1
(3) POINT1, POINT2のところを0C9FEH(DEC CL)　…… 音のアップ変化2
(4) POINT1, POINT2のところを0C1FEH(INC CL)　…… 音のダウン変化2

どんな音になりましたか．インベーダーの襲来を思わせるような，そんなカッコイイ音になった方もいるかもしれません．同じデータでも効果音のコール先を4種類変えて実験してみると，色々な音に変化するはずです．データによっては，かなり面白い音が作れると思います．ただし，こういう特殊音にはデータ表などありませんから，すべて自分で記録管理しておかないと，イザというときに毎回テストの連続ということになってしまいます．その辺は自分なりに工夫してください．

リスト4-2　BEEPによる効果音(LIST 4-2.ASM)

```
;
;****** List 4-2 ******
;
SND1:   ;SouND 1
        MOV     AX,ØC9FEH          AXに(DEC CL)のマシン語コードを代入
        JMP     CPT2               CPT 2へジャンプ
SND2:   ;SouND 2
        MOV     AX,ØC1FEH          AXに(INC CL)のマシン語コードを代入
        JMP     CPT2               CPT 2へジャンプ
SND3:   ;SouND 3
        MOV     AX,ØC9FEH          AXに(DEC CL)のマシン語コードを代入
        JMP     CPT12              CPT 12へジャンプ
SND4:   ;SouND 4
        MOV     AX,ØC1FEH          AXに(INC CL)のマシン語コードを代入
;
CPT12:
        MOV     DI,offset POINT1   DIにPOINT1のオフセット・アドレスを代入
        MOV     [DI],AX            POINT1のマシン語コードを書き換える
CPT2:
        MOV     DI,offset POINT2   DIにPOINT2のオフセット・アドレスを代入
        MOV     [DI],AX            POINT2のマシン語コードを書き換える
        CALL    MUSIC              音楽演奏
        MOV     AX,9Ø9ØH           NOPコードを2つ分AXへ格納
        MOV     DI,offset POINT1   POINT1のオフセットをDIへ
        MOV     [DI],AX            POINT1をNOPに初期化
        MOV     DI,offset POINT2   POINT2のオフセットをDIへ
        MOV     [DI],AX            POINT2をNOPに初期化
        RET                        リターン

        include LIST4-1.ASM        LIST4-1.ASMを取り込む
```

テスト4-2　テスト・プログラム(TEST 4-2.ASM)

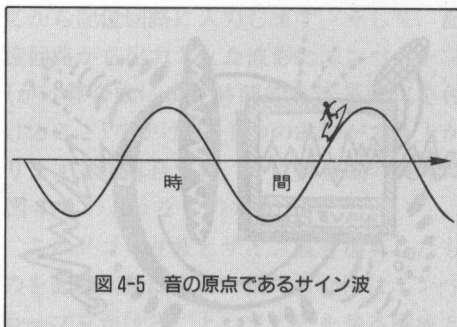
```
;****** test 4-2-T ******

CODE    segment                              命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,SS:STSEG

PTEST:  ;Program TEST
        MOV     AX,CS                        AX ← CS
        MOV     DS,AX                        AX レジスタを介して DS に CS を格納
        MOV     BX,offset MDTOP              音楽データの先頭アドレス
        CALL    SND1                         効果音1を出す
        MOV     AL,Ø                         正常終了
        MOV     AH,Ø4CH                      プロセスの終了
        INT     21H                          ファンクションコール

        include LIST4-2.ASM                  LIST4-2.ASM を取り込む

CODE    ends                                 CODE と名付けたセグメントの終わり

STSEG   segment STACK                        スタックセグメントの宣言
        db      1ØØH dup (?)                 メモリ領域を100H バイト確保
STSEG   ends                                 スタックセグメントの終わり

        end                                  プログラム・エンド
```

4. FM音源とSSG … FM音源ボード専用

本節と次の5節は，PC-9801シリーズ用のFM音源ボードをお持ちでない方は，実際にテストをすることはできません．しかし，将来のためにいちおう読んでおいても損にはならないと思いますが，そのあたりの判断はオマカセいたします．

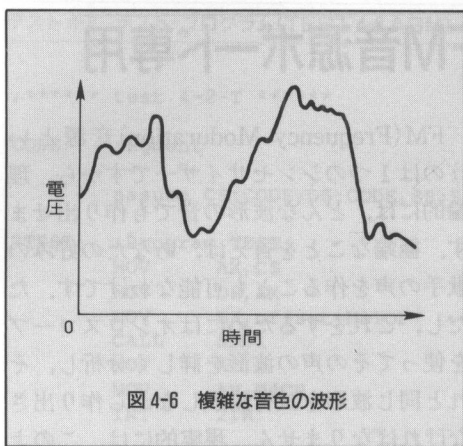
音は空気の震動ですから，その振動波の間には，かならず変化に要する時間があるわけです．したがって，本当の音の原点とは図4-5にあるようなサイン波のことをいいます．

しかし，このようなキレイなサイン・カーブだけでは世のなかにあるさまざまな音を表現することはできません．世のなかの音というものはさまざまなノイズがはいって1つの音となっているわけです．そのような音を電気的に作り出すには，この基本のサイン波に色々な細工をして，似たような音の波に加工しなければなりません．これを疑似的にできるのがSSG(PSG)音源であり，本格的にできるのがFM音源なのです．

図4-5　音の原点であるサイン波

FM(Frequency Moduration)音源というのは1つのシンセサイザーですから，理論的には，どんな波形の音でも作り出せます．極端なことを言えば，あなたの好みの歌手の声を作ることも可能なわけです．ただし，これをするためにはオシロスコープを使ってその声の波形を詳しく分析し，それと同じ波形を試行錯誤しながら作り出さなければなりません．現実的には，このようなことがだれにでもできるものではありませんし，偶然に期待するしか方法はなさそうです．

それでは，せっかくのFM音源がまるで宝の持ち腐れになってしまうのではないか，ということになりますが，そのために最初からメーカー側で，色々な楽器の音や効果音を用意してくれているのです．BASICで音色番号を指定すれば，ピアノや小鳥のさえずり音が簡単に出てくるのはそのためです．

これに対し，SSG(Synthesized Sound Generator)というのは1つのプリセットされた音色とノイズで構成されており，音色の決まった音楽演奏はできても，効果音の用意などはありません．そのため，効果音を出すには音源とノイズを組み合わせたり，エンベロープ(時間的な音量変化)をかけるなどして，力技で作ることになります．ですから，FM音源のようにどんな音でもできるというわけにはいきませんが，それでも工夫次第でいろいろな効果音を作ることが可能です．

図 4-6　複雑な音色の波形

このような特徴のあるFM音源とSSGを両方兼ね備えたサウンド・ボードですが，ここでその使用方法すべてを書くことはスペース的に無理です．それだけで，それこそ1冊の本になってしまいます．そこで，ここではFM音源の基本的な原理と，マシン語によるFM音源コントロールについて，とにかくドレミを出すということを最終目標にすることにしました．

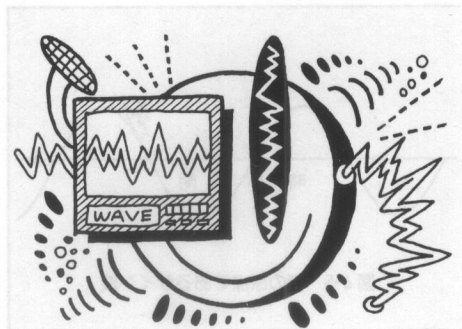
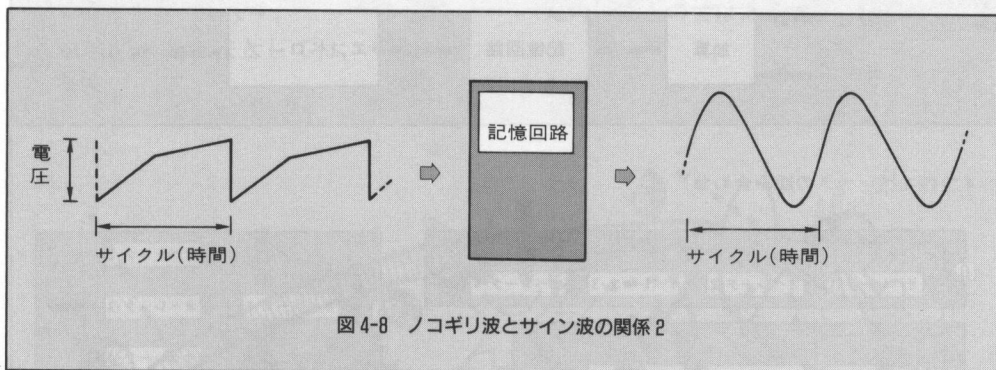
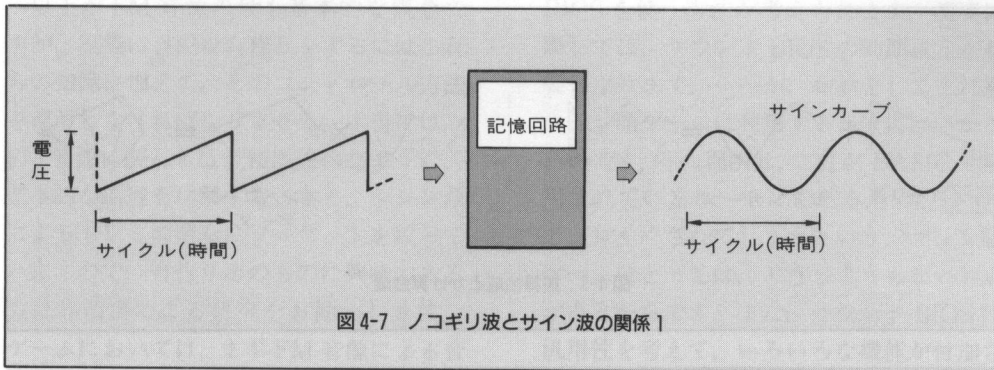
まず，スピーカーに送る信号ですが，音色が加わることによりビープ音楽よりずっと波形が複雑になります．これは時間の変化とともに複雑な形の電圧変化が加わるということです．

この図4-6にある波形は単なる例であって特別な意味はありませんが，このような複雑な波形をどのようにして作り出していくのか，を解明しなければなりません．そのためにはFM音源の基礎となっているノコギリ波とサイン波の関係を理解する必要があります．というのは，FM音源ではすべての音を2つの波形の合成という形で作っており，その基本となるのがノコギリ波とサイン波であるからです．そして，どんな複雑な波形もこの合成を何度も繰り返すことによって，作成可能となるのです．このような合成の基本となるのは，簡単に言うと図4-7に示すようにノコギリ波から生み出されるサイン波にあります．

図4-7に示した記憶回路のなかには，サイン波の素がはいっています．このサイン波の素というのは，はいってくる電圧によって出る電圧を決定するもので，図4-7のように直線的に上昇する電圧のときに，きれいなサイン・カーブを描きます．したがって，出てくるサイン波の周波数を決めているのはノコギリ波であり，ノコギリ波の周波数がそのままサイン波の周波数になるのです．また，ノコギリ波の電圧変化率(数学的に言うと傾き)を途中で変えると，出てくるサイン波の形も途中で変わってきます．たとえば図4-8ではノコギリ波の電圧が前半は急激に，後半は穏やかに上昇しているため，出てくるサイン波は標準形とは違って前半のカーブが急になっています．

このような関係から，記憶回路に入れる

図 4-7　ノコギリ波とサイン波の関係 1

図 4-8　ノコギリ波とサイン波の関係 2

ノコギリ波をより複雑にすれば，出てくる波形はさらに複雑に変化することが想像できます．そのため，実際にはノコギリ波とサイン波を先に加算合成し，複雑な形にしてから記憶回路に入力します．そして，記憶回路から出力された波形にエンベロープ(かけ算合成により時間的な音量変化を付けること)をかけて，1つの波形ができ上がります．加算合成と，かけ算合成の実体は図 4-9 のようなものです．

このノコギリ波とサイン波を加算したものを記憶回路に入れ，出てきた波にエンベロープをかける，ということを基本構成としたものを**オペレータ**といいます．FM 音源ボードに搭載されているシンセサイザー IC は，1つの音に対して4つのオペレータを持っており，図 4-10 にあるような組み合せで，さらに複雑な波形を出力するようになっています．これらのオペレータの内，いちばん最後にあって音の波形を出力するようになっているものを**キャリア**といい，キャリアに対して変調用の波形を送るオペレータのことを**モジュレータ**といいます．また，このようなオペレータの組み合せ(接続の仕方)のことを**アルゴリズム**といいます．

ノコギリ波
サインカーブ
エンベロープ
データ

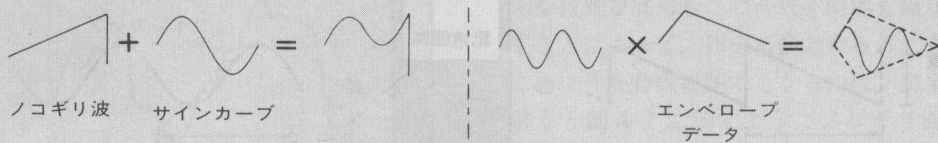



図 4-9　加算合成とかけ算合成

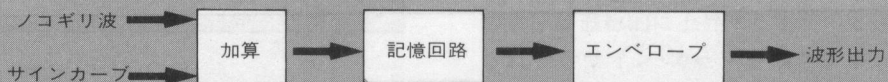
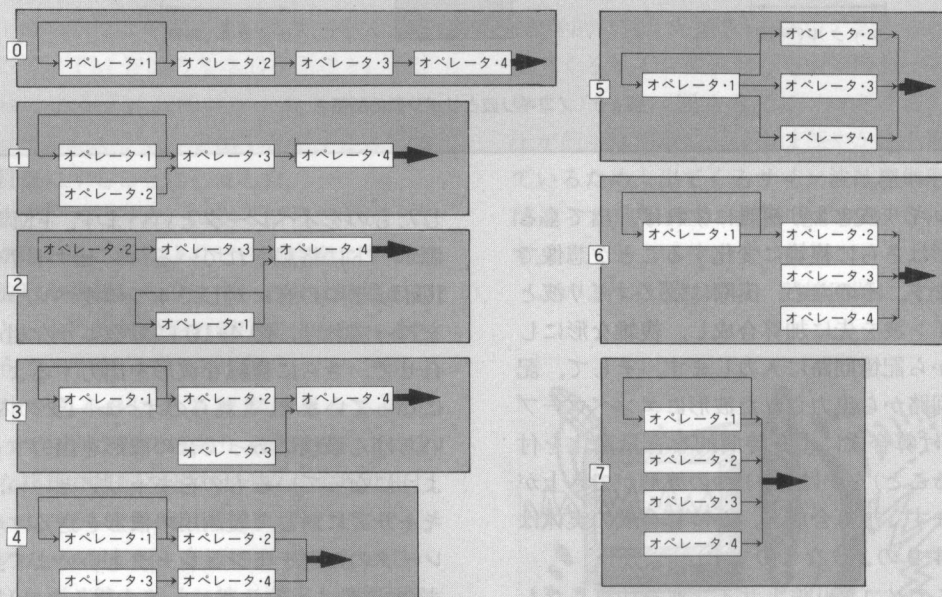



図 4-10　オペレータの仕組みと組み合わせ

以上がFM音源の最も基本的な概念ですが，実際に自由な音作りをするにはこれらの知識に加えて，そのコントロール方法を理解しなければなりません．本書では，FM音源に関しては最初に述べたように，基本的な原理を理解することと，マシン語によるドレミの演奏にターゲットを絞っていますので，音作りそのものに興味のある方は専門書による研究をお勧めします．ゲームにおいては，まずFM音源による音楽演奏を，マシン語レベルで利用できるようになることが第一です．

比較的簡単な方法としては，サウンドBIOSを使う方法が考えられます．演奏に際しては，サウンドBIOSの初期設定が必要となります．……が，結論としてこれをマシン語ゲームに利用するわけにはいかないのです．その理由は，これがBASICで使用されているルーチンであるため，STOPでブレイクされてしまうという，マシン語ゲームにとっては，どうしようもない欠点があるからです．また，サウンドBIOSは汎用性を考えて，いろいろな機能が付加されているために，処理スピードの点でも満足のいくものではありません．

5. ミュージック … FM音源でハープシコード

サウンド BIOS に頼れないとなると，シンセサイザーIC(YAMAHA の YM-2203)を自分で直接コントロールするしかありません．つまり，このシンセサイザーIC の FM 音源部(以下, FM 音源 IC と略す)に対して，音色とか音階のデータを送るということです．難しそうな気がするかもしれませんが，そこには簡単なルールがありますから，素直にそれに従いさえすればいいのです．

では，まずはルール1として，音を出すための大まかな手順を理解してください．

1)　FM 音源 IC に対し, 音色データを送る
2)　FM 音源 IC に対し, 音程データを送る
3)　FM 音源 IC に対し, キー・オン(音を出すという合図)データを送る
　………………音がでる……………………
4)　音の長さ分だけ，ウェイトをおく
5)　FM 音源 IC に対し, キー・オフ(音を止めるという合図)データを送る
　………………音が止まる………………

音色データは，音色の変更がない限りデータを送り直す必要はありませんから，実際に曲を演奏する場合は 2)～5) を繰り返すということになります．いかにも簡単そうなルールの1ですが, 問題はこの「データを送る」という部分にあります. ウェイト以外はすべて「データを送る」になっていますが, FM 音源 IC のどこに，どんなデータを，どういう方法で，ということが具体的にまったく明記されていません．実は，これが本節最大のヤマ場であるルールの2というわけなのです．テーマが色々ありますから，1つひとつ順を追ってクリアしていくことにしましょう．

FM 音源 IC のアドレス・マップ

まず，データの送り先ですが，FM 音源 IC のレジスタアドレスは図 4-11 のようになっており，コメントのある部分のアドレスが，ここで音を出すために必要なレジスタ(FM 音源 IC のメモリ)です．それ以外のレジスタに関しては，ここでは無視します．

この図を見ると，1つのアドレスで複数のパラメータを持っているものがあり，あまりわかりやすい構成とはいえません．しかし, ルール1にあるデータの内, 音程データの送り先とキーのオン／オフは，確認できたと思います．そうすると，必然的に残りのレジスタ(コメントのないものは除く)は，音色に関するレジスタということになります．つまり，めんどうなのは最初に音色を設定することで，音階を演奏するのは3つのレジスタにデータを送るだけでいいのです. ただし, FM 音源部は全部で3チャンネルありますから，3重奏をするためには，データもチャンネルごとに別々に送ら

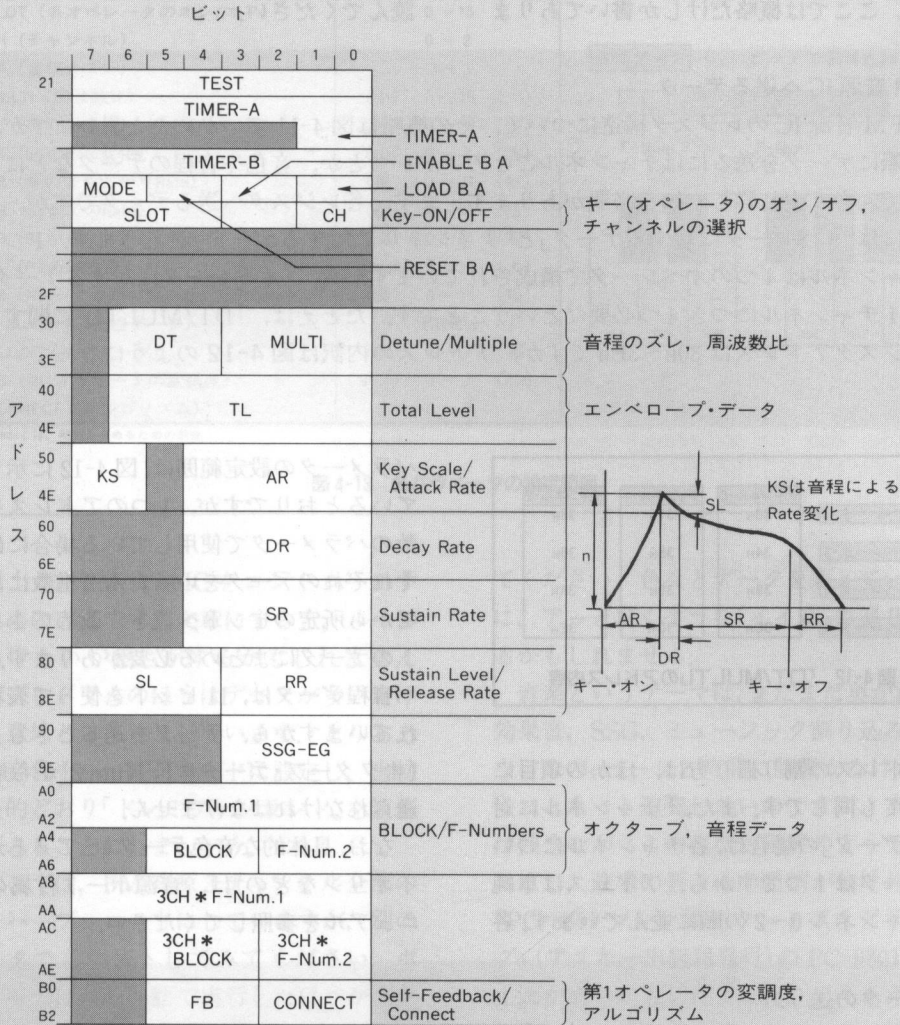



図 4-11　FM音源ICのレジスタアドレス

なければなりません.

なお,各レジスタ・アドレスの意味については,ここでは概略だけしか書いてありませんので,詳しくは色々な参考書や,サウンドボードについているマニュアルなどを読んでください.

FM 音源 IC へ送るデータ

FM 音源 IC のレジスタ構造について,その概略は図 4-11 でつかめたと思いますが,実際にデータを送るにはチャンネルごとのアドレスとか,音色や音程のデータなどについて,もう少し詳しく知る必要があります.まず,各レジスタへ送るデータですが,これには「オペレータに対するデータ」と「チャンネルに対するデータ」があります.1つのチャンネルは4つのオペレータで構成されていますから,「オペレータに対するデータ」は1チャンネルにつき4つ必要だということです.たとえば,『DT/MULTI』に関するレジスタアドレスは 30_H〜$3E_H$ ですが,アドレスの内訳は図 4-12 のようになっています.

図 4-12 『DT/MULTI』のアドレス内容

アドレスの割り振り方は,ほかの項目についても同じです.また,「チャンネルに対するデータ」の場合は,各チャンネルについてデータは1つですから,アドレスは単純にチャンネル0〜2の順に並んでいます.各パラメータの設定範囲は,図 4-12 に示されているとおりですが,1つのアドレスを複数のパラメータで使用している場合には,それぞれのデータをいったん2進数に直してから所定のビットへ置き,改めて1バイトのデータにまとめる必要があります.

音程データは,11 ビットを使って表現されていますから,データを送るときも上位(オクターブ・データ+F-Num.2)から順に送られなければなりません.

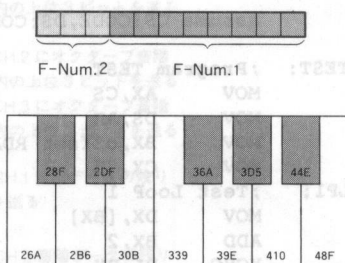
なお,具体的な音色データ(ピアノとかバイオリンなどの)は,音源ボード付属のマニュアルを参照してください.

データの送り方

たくさんのレジスタアドレスを持った FM 音源 IC ですが,本体のメイン CPU とは I/O ポートの 188_H と $18A_H$ だけで結ばれています.そのため,FM 音源 IC へデータを送るには,出力ポート 188_H へまず送り先(レジスタアドレス)を出力し,次に出力ポート $18A_H$ へデータを出力する,という方法をとっています.なお,各ポートへデータを出力した後は,LSI 内部での処理が終るまでウェイトを置かなければなりません.

パラメータ	設定範囲		
SLOT（各オペレータのオン/オフ）		0～15	
CH（チャンネル）		0～2	
DT（音程のズレ）		−3～3	
MULTI（周波数比）	（小）	0～15	（大）
TL（出力レベル）	（大）	0～127	（小）
KS（EGのRate変化）	（小）	0～3	（大）
AR（音の立ち上がりRate）	（長）	0～31	（短）
DR（SLまでのRate）	（長）	0～31	（短）
SR（SLからTL＝0までのRate）	（長）	0～31	（短）
SL（DRの減衰量）	（小）	0～15	（大）
RR（キー・オフからTL＝0までのRate）	（長）	0～15	（短）
F-Num.2・1（音程データ）		〔右図〕	
BLOCK（オクターブ）	（低）	0～7	（高）
FB（第1オペレータの変調度）	（オフ）	0～7	（4π）
CONNECT（アルゴリズム）		0～7	

＊Rateとは，時間を決めるための割合

図4-13 各パラメータの設定範囲

以上が，FM音源ICをコントロールするための方法です．文章にすると，ルールだらけという感じがするかもしれませんが，プログラムのほうはデータが多いだけで，意外とスッキリしています．とりあえず，音色をハープシコードに設定し，当初の目的どおり「ドレミファソラシド」を演奏してみることにしましょう．

ここでは，FM音源3チャンネル全部の音をハープシコードにしていますが，別のデータでもテストしてみてください．ボリュームはマシン語で実行したほうが大きくなります．これはBASICにおけるボリュームの設定が少し低めになっているためです．また，いずれにしても音色を自分で直接コントロールできるようになったわけですから，メーカー指定の音色にこだわらずに，よりリアルな音を求めていってみてください．色々とデータを変えている内に，アッと驚くような「イイ音」に巡り会えるかもしれません．

音楽というテーマは，まだまだ奥が深く，効果音，SSG，ミュージック割り込み……などなど，ページと時間に制限がなければ，もっともっと追究したいのですが，本書では残念ながらこの程度が限界のようです．ただし，読者の皆さんの希望が強ければ，『PC-8801マシン語サウンドプログラミング』(アスキー出版局発行)のPC-9801版の企画が立つかもしれません．とりあえず，本書はゲームマシン語のテーマが，まだまだ終わっていませんから(というより，先のほうが長い)，ここはFM音源に対する自信だけを付けたことにして，先へと進むことにいたしましょう．

リスト 4-3　FM音源によるハープシコード演奏(LIST 4-3.ASM)

```
;****** LIST4-3 *****

CODE    segment                         命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,SS:STSEG
;
PTEST:  ;Program TEST
        MOV     AX,CS
        MOV     DS,AX
        MOV     BX,offset RDATA
        MOV     CX,75                   音色の設定(ハープシコード)
TLP1:   ;Test LooP 1
        MOV     DX,[BX]                 DH←FM音源IC内部レジスタ番号
        ADD     BX,2                    DL←上記レジスタに送るデータ
        XCHG    DL,DH
        CALL    OUTREG
        LOOP    TLP1
;
        MOV     BX,offset SDATA         BX←音楽データの先頭アドレス
        MOV     CX,8                    CX←演奏される音符数
TLP2:   ;Test LooP 2
        CALL    PLAY                    音を出す
        CALL    PWAIT                   ウェイトを置く
        CALL    KEYOFF                  音を止める
        LOOP    TLP2                    上記をCX回繰り返す
        MOV     AX,ØCØØH                キーボード・バッファ・クリア
        INT     21H                     ノーマル・エンド
        MOV     AL,Ø                    プロセスの終了
        MOV     AH,4CH                  ファンクションコール(MS-DOSへ戻る)
        INT     21H
;
OUTREG: ;OUT REGster
        PUSH    CX                      CXレジスタ保存
        MOV     CX,DX                   CX←レジスタ番号&データ
        MOV     DX,188H                 DX←ポート番号セット…… 188H
REDST:  ;REaD STatus
        IN      AL,DX
        AND     AL,8ØH                  入力ポート188Hのビット7が0まで待つ
        JNE     REDST
        MOV     AL,CH
        OUT     DX,AL                   FM音源IC内部レジスタの選択 …… ①
        rept    24
        NOP                             無駄命令によるウェイト
        endm                            (20MHzで85クロック以上確保)
        MOV     DX,18AH
        MOV     AL,CL                   ①で選択したレジスタへデータを送る
        OUT     DX,AL
        POP     CX                      CXレジスタ復元
        RET                             リターン
```

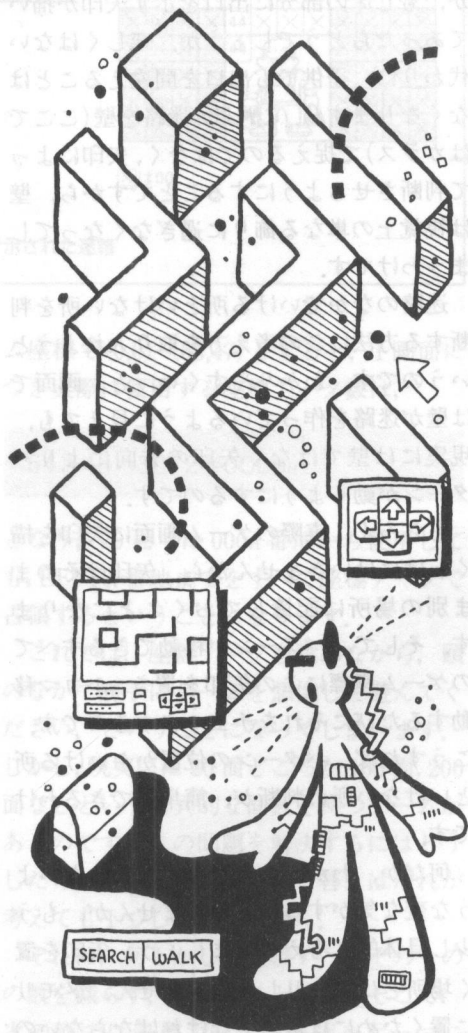
```
;
PLAY:   ;PLAY music
;
        MOV     DL,[BX]
        INC     BX          FM音源CH.1にオクターブ音階
        MOV     DH,ØA4H     データの内の上位3ビットを送る
        CALL    OUTREG
        INC     DH          FM音源CH.2にオクターブ音階
        CALL    OUTREG      データの内の上位3ビットを送る
        INC     DH          FM音源CH.3にオクターブ音階
        CALL    OUTREG      データの内の上位3ビットを送る
        MOV     DL,[BX]
        INC     BX          FM音源CH.1音階データの残り
        MOV     DH,ØAØH     8ビットを送る
        CALL    OUTREG
        INC     DH          FM音源CH.2音階データの残り
        CALL    OUTREG      8ビットを送る
        INC     DH          FM音源CH.3音階データの残り
        CALL    OUTREG      8ビットを送る
;
        MOV     DX,28FØH    FM音源CH.1のキー・オン
        CALL    OUTREG
        MOV     DX,28F1H    FM音源CH.2のキー・オン
        CALL    OUTREG
        MOV     DX,28F2H    FM音源CH.3のキー・オン
        CALL    OUTREG
;
PWAIT:  ;Program WAIT
        PUSH    CX
        MOV     CX,8ØØØH
WTLP:   ;WaiT LooP          無駄命令によるウェイト
        PUSH    AX          …… 音の長さとなるCXの値によって長さが変化する
        POP     AX
        LOOP    WTLP
        POP     CX
        RET
;
KEYOFF: ;KEY OFF
        MOV     DX,28ØØH    FM音源CH.1のキー・オフ
        CALL    OUTREG
        MOV     DX,28Ø1H    FM音源CH.2のキー・オフ
        CALL    OUTREG
        MOV     DX,28Ø2H    FM音源CH.3のキー・オフ
        CALL    OUTREG
        RET
;
RDATA:  ;Register DATA
                            Detune/Mutipleのレジスタ・アドレスと送るデータ
        db      3ØH,12,31H,12,32H,12
        db      38H,1ØH+15,39H,1ØH+15,3AH,1ØH+15
```

```
        db      34H,1,35H,1,36H,1
        db      3CH,70H+3,3DH,70H+3,3EH,70H+3
                                                    Total Level のレジスタ・アドレスと送るデータ
        db      40H,32,41H,32,42H,32
        db      48H,57,49H,57,4AH,57
        db      44H,30,45H,30,46H,30
        db      4CH, 0,4DH, 0,4EH, 0
                                                    Key Scale/Attack Rate のレジスタ・アドレスと送るデータ
        db      50H,31,51H,31,52H,31
        db      58H,0C0H+31,59H,0C0H+31,5AH,0C0H+31
        db      54H,31,55H,31,56H,31
        db      5CH,80H+31,5DH,80H+31,5EH,80H+31
                                                    Decay Rate のレジスタ・アドレスと送るデータ
        db      60H,12,61H,12,62H,12
        db      68H,2,69H,2,6AH,2
        db      64H,12,65H,12,66H,12
        db      6CH,133,6DH,133,6EH,133
                                                    Sustain Rate のレジスタ・アドレスと送るデータ
        db      70H,4,71H,4,72H,4
        db      78H,4,79H,4,7AH,4
        db      74H,4,75H,4,76H,4
        db      7CH,7,7DH,7,7EH,7
                                                    Sustain Level/Release Rate のレジスタ・アドレスと送るデータ
        db      80H,10H+10,81H,10H+10,82H,10H+10
        db      88H,0F0H+6,89H,0F0H+6,8AH,0F0H+6
        db      84H,6,85H,6,86H,6
        db      8CH,20H+7,8DH,20H+7,8EH,20H+7
                                                    Self-Feedback/Connection のレジスタ・アドレスと送るデータ
        db      0B0H,38H+2,0B1H,38H+2,0B2H,38H+2
;
SDATA:  ;Sound DATA
        db      18H+2,6AH                           4オクターブ目ド
        db      18H+2,0B6H                          4オクターブ目レ
        db      18H+3,0BH                           4オクターブ目ミ
        db      18H+3,39H                           4オクターブ目ファ
        db      18H+3,9EH                           4オクターブ目ソ
        db      18H+4,10H                           4オクターブ目ラ
        db      18H+4,8FH                           4オクターブ目シ
        db      20H+2,6AH                           5オクターブ目ド

CODE    ends                                        命令の置かれているセグメントの終わり

STSEG   segment stack                               スタック用セグメントの開始
        db      100H dup(?)                         データ域を 100H バイト確保
STSEG   ends                                        スタック用セグメントの終わり

        end                                         プログラム・エンド
```

5章

●迷路型ゲーム

●リアルタイムなアクションゲームを分類すると，シューティングゲーム(インベーダー型)，迷路型ゲーム(パックマン型)，スクロールゲーム(ゼビウス型)の３つに大別できます．なかでもアクションゲームは，ほとんどのものが何らかの迷路と切っても切れない関係と言えるかもしれません．一部には食傷気味という声も聞こえますが，まだまだその人気は落ちてこないようです．

●本来の意味での迷路とは，出口を求めてウロウロするような道のことを言うのですが，ゲームの場合にはもう少し拡大解釈し，画面のなかに行けるところと行けないところがあるようなものを，迷路型ゲームと言ってます．本章では，この迷路の秘密を全部バラして，迷路をただの道にしてしまいましょう．

1. 座標データ…いける? いけない?

よく遊園地などに，ガラスで囲まれた部屋がたくさんある迷路があります．つまりどこが出口かわからなくして楽しむのですが，もし床の部分に出口を示す矢印が描いてあったらどうでしょうか．楽しくはない代わりに，子供でも出口を間違えることはなくなりますね．これは，迷路を壁(ここではガラス)で捉えるのではなく，矢印によって判断させるようにすることですから，壁は視覚上の単なる飾りに過ぎなくなってしまうわけです．

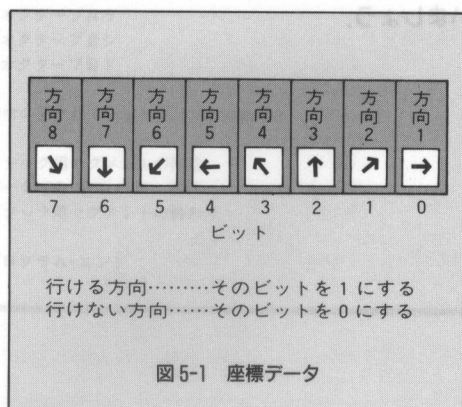
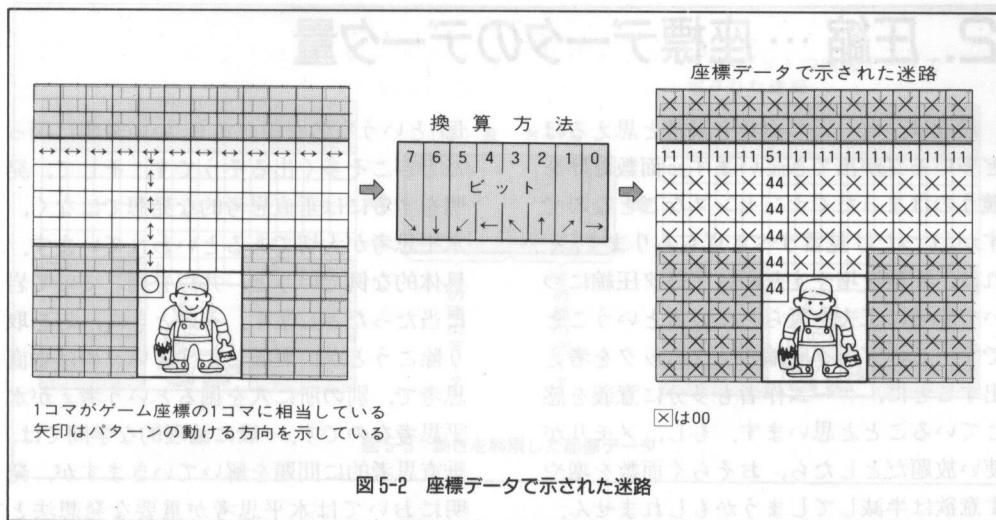
迷路のなかでいける所といけない所を判断する方法にこの考え方を取り入れようというのです．わかりやすくいうと，画面では壁が迷路を作っているように見えても，現実には壁ではなく矢印の方向によりパターンが動くようにするのです．

もちろん，実際のゲーム画面に矢印を描くわけにはいきませんから，矢印はそのまま別の場所に記憶しておくことになります．そして，パターンが移動できるすべてのゲーム座標にこの矢印を置き，1コマ移動するたびにそれをチェックするのです．こうすれば，パターンの位置からいける所といけない所の判断が，簡単にできるわけです．

何だか，わかったような，わからないような変な気がするかもしれませんが，もう少し具体的に考えてみましょう．矢印を置く場所とはメモリしかありません．メモリに置くためには数値でなければならないのは当然のことです．そこで，まず矢印を数値に変換する必要が生じてきます．矢印とは，すなわちいける方向のことですから，全部で8方向分あります．いっぽう，メモリに置ける数値も，1バイトつまり8ビットですから，1ビットごとに1つの方向を表すようにし，いける場合は1，いけない場合は0とすれば，そこの座標の矢印をすべて表現できることになります．このようにして数値に変換された矢印のことを，その座標が持っているデータということで，座標データと呼ぶことにします．そして，ここでは，矢印と座標データとの関係を図5-1のように設定して，矢印を数値に変換しています．

この座標データは，パターンを移動させる前に読み出して，その方向にあるビットをチェックすればいいだけですから，利用方法も非常に簡単で確実です．また，どんなに複雑な迷路でも，これさえあれば作れるのですから迷路の必需品ともいえます．

図5-1 座標データ

図 5-2　座標データで示された迷路

では，図 5-2 を見てください．

この座標データには，このような方法以外にも色々な作り方が考えられます．たとえば，壁の部分を 0，道の部分を 1，ハシゴは 2，道に金塊が置いてあれば 3……というようなデータでも迷路の判断ができるわけです．つまり，座標データの内容に関しては，利用するあなたのアイデア次第ということであり，図 5-2 に示した座標データはそのなかの 1 つの例に過ぎません．いずれにしても，迷路の判断には画面とは別に何らかの座標データを持つ必要があるのです．

一般に，どんなゲームであっても画面数が少ないと，面白さも欠けているように思われる傾向があります．そのため，最近は最低でも 20 面くらいの画面数が要求されますが，そのような場合このままの座標データでは，メモリが足りなくなってしまいます．たとえば，迷路画面のサイズをゲーム座標で(0,0)－(63,47)とすると，1 画面につき実際に使用する座標データ数は，

$$64\times48=3072=0C00_H$$

となり，20 面では 000H 番地から使用しても，EFFFH 番地までをすべて座標データで占領するということになります．

これでは，座標データを眺めながら，頭のなかで勝手にゲームを想像して遊んでください，ということになってしまいます．しかし，現実には 20 面どころか 100 面，200 面などという驚異的な面数を持つゲームがあるのです．この問題を解決するにはどうしたらいいか，それに対する答えはだれが考えても 1 つしかありません．

それは，何とかして座標データそのものの数を減らす，ということです．そこで，次の節では座標データの圧縮をテーマにし，その可能性を追求してみましょう．

2. 圧縮…座標データのデータ量

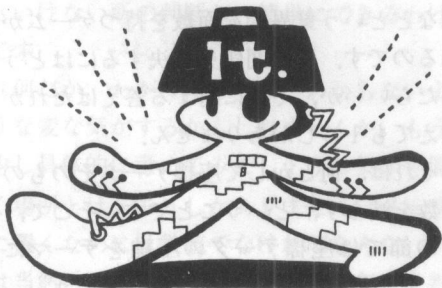
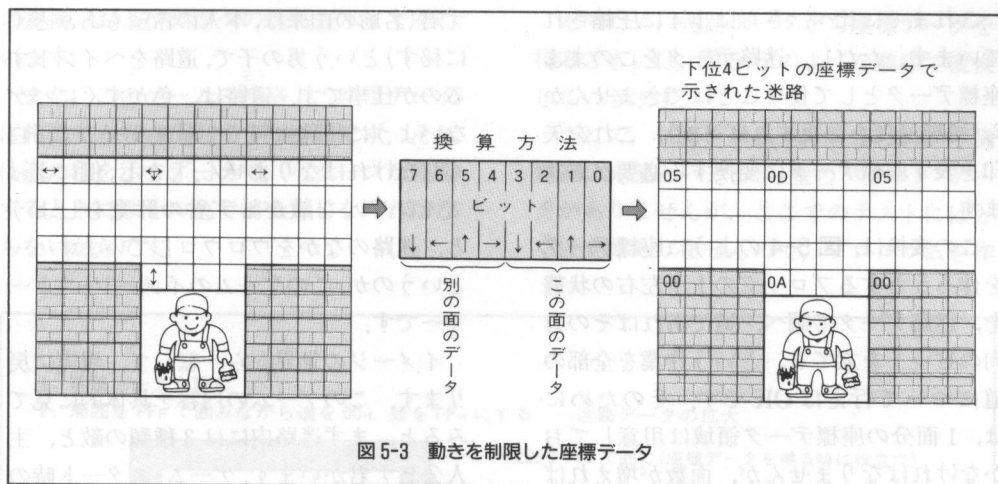
最近は，ゲームの面数も異常と思えるほど多いモノが出てきています．面数だけを競うのはまったくナンセンスなことなのですが，1つだけ見習うべき点もあります．それは，面数を増やすためにデータ圧縮について非常に工夫を凝らしているということです．このデータ圧縮のテクニックを考え出すことに，ゲーム作者も多分に意義を感じていることと思います．もし，メモリが使い放題だとしたら，おそらく面数を増やす意欲は半減してしまうかもしれません．もっとも，最近では，PC-9801が世に出た頃と比べると使い放題といっても過言ではありませんが．

あまり使い過ぎると，プログラムの効率や，ディスクの枚数などにも影響してきますので，メモリを有効に使うためには，やはりデータを圧縮することを考えなければならなくなってきます．

圧縮というのは，必要があって初めて出てくるテーマですから，どこか発明に似たところがあります．発明には「必要は発明の母」という有名な諺があり，いい発明は困ったときこそ多く出るそうです．そして，発明をするには垂直思考的な発想ではなく，水平思考が大切であるといわれています．具体的な例でいうと，井戸を掘っていて岩に当たったとします．そのときに，岩を取り除こうとか，爆破しようというのは垂直思考で，別の所に穴を掘るという考えが水平思考なのです．一般に論理的な学問では，垂直思考的に問題を解いていきますが，発明においては水平思考が重要な発想法となっているそうです．

さて，問題になっている座標データ圧縮の方法ですが，ここでは2段階に分けて圧縮をかけてみます．まず第一段階では，パターンの方向変更は4コマごとということにして，座標データの数そのものを1/16にいきなり減らしてしまいます．4コマごとに方向変更を制限するといっても，反対方向への変更はつねに可能ですから4コマを1ブロックとするキチンとした迷路であれば，制限がないのとまったく同じことです．ただし，広場のような部分がある場合には，方向変更が4コマごとにしかできないため，多少動きがギクシャクします．

次に，パターンの斜め移動を禁止して，上下左右だけの動きに制限します．これで，1つの座標データを上位4ビットと下位4ビットに分けて別の面で使うことができるようになります．その結果，初期のものに比べると1/32のデータ量で済むことになったわけです．

図 5-3　動きを制限した座標データ

上の図 5-3 を見ると，データ量は確かに大幅に少なくなっています．この方式による座標データであれば，200 画面の迷路を作ることが可能であることも間違いありません．……が，これは実際には「圧縮されたデータ」とは言えないのです．というのは，この座標データはパターンの動きに制限を加えた結果生まれたものであり，データとしての基本的な構造は，図 5-2 と何ら変わりがないからです．圧縮されたデータというのは本来，プログラムによりもとの状態に戻すことができるはずですが，図 5-3 のデータはこれ自身がデータであって，もとに戻すべき姿はありません．

そこで，方向変更は 4 コマごとという条件は同じにして，次のような考え方で本格的な圧縮をかけてみます．なお，4 コマごとの方向変更ということは，壁，道，移動パターンをすべてゲーム座標で 4×4 コマのサイズに統一する，ということが前提となります．

1. 画面で壁の部分を 1 とする．
2. 画面で道の部分を 0 とする．
3. 横 8 ブロック(1 ブロックはゲーム座標で 4×4 コマとする)分の壁と道を連続する 8 ビット(1 バイト)のデータとみなす．

この方法により作られたデータは，座標データではなく迷路の状態(壁か道)を表しているので迷路データといえます．ゲーム座標で(0,0)-(63,47)の画面サイズを例にとって，図 5-3 と必要バイト数を比較してみましょう．

迷路データによる必要バイト数　……　64/4/8×48/4=24
図 5-3 によるバイト数　……　64/4/2×48/4=96

これまでに比べ，さらに1/4に圧縮されています．ただし，迷路データをこのまま座標データとして使うことはできませんから(処理速度を無視すれば可能)，これを矢印を表す座標データに変換する必要があります．

この変換は，図5-4のように座標データを作ろうとするブロックの上下左右の状態を，迷路データで調べ，道であればその方向のビットを立てる，という作業を全部の道について行えばOKです．そのためには，1面分の座標データ領域は用意しておかなければなりませんが，面数が増えれば増えるほど，このデータ圧縮の効果も大きくなってきます．

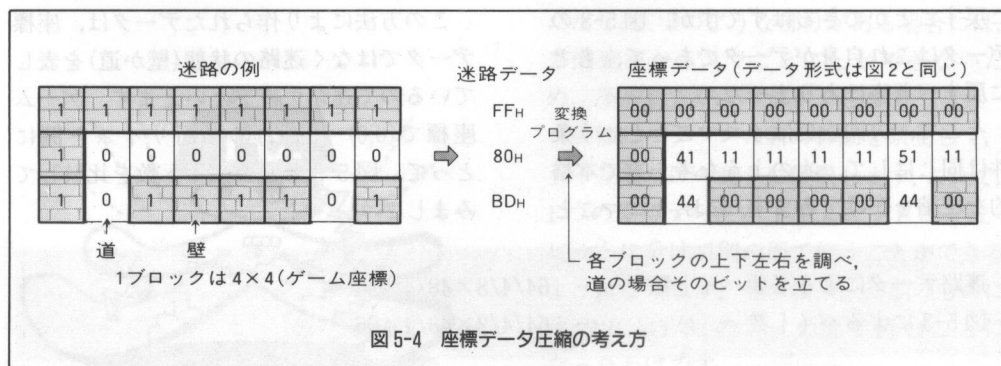
なお，斜め移動はここでは無視していますが，上下左右に加えて斜めの状態も調べればできるようになります．

さて，データ圧縮についての基本的な方針が理解できたところで，この5章で作る迷路ゲームに駒を進めることにしましょう．

今回は，ゲームにも〝ペンキ・ボーイ〟という名前を付けてみました．主人公は，ヒデ君(名前の由来は，本人の希望もありとくに秘す)という男の子で，道路をペイントするのが仕事です．道路は，色がすぐにハゲないように特殊塗料で7層塗りをして，白くしなければなりません．しかし，例によってイジワルな敵がヒデ君の邪魔をしようと，迷路のなかをウロウロしています……というのが，このゲームのイメージ・ストーリーです．

イメージの世界から一転して，現実に戻ります．このゲームの内容を具体的に見てみると，まず迷路内には3種類の敵と，主人公ヒデ君がいます．ゲーム・スタート時の道の色は黒ですが，ヒデ君が歩いた跡は青(1)から白(7)へパレット番号順に変化していきます．そして，すべての道が同一色になるとボーナス点がはいり，すべての道が白になった時点でゲーム終了となります．SPACEを押すと，道の色を変化させずに歩くことができ，また敵と衝突しても死なないものとします．

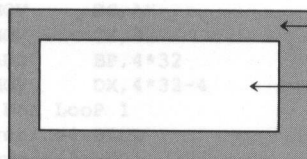
作成する画面は，練習ですから1面だけしかありません．以上がゲームの概略ですが，本節では迷路データから座標データへ

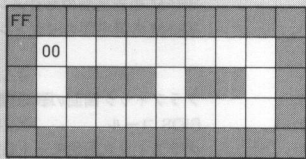
図5-4 座標データ圧縮の考え方

の変換，および画面への迷路表示までを行います．

また，このゲームの性格上，座標データに相当するものがもう1つ必要です．それは道の色を示すデータで，パターンが移動するときにはその色で消去をしなければならないからです．

そのため，迷路データから座標データを得るときには，図5-5のような順序で変換していきます．

これで，プログラムへはいるための予備知識は完璧といえます．壁のパターン・データがありませんが，ここでのテストにはとくに支障ありませんので，プログラムを作

1．迷路データを読む
2．周囲を FFH で囲みながら道を 00H，壁を FFH にする．…迷路データの拡大

3．作成されたデータ（周囲の FFH は除く）から，座標データを作る。道の上下左右をチェックし壁の場合は，行ける方向がないので座標データを 00H とする．…座標データの作成
4．迷路データから周囲の FFH を取り除き，連続したデータとする。このデータが，道の色を示すデータ・エリアとなり，道の色が変化するごとにデータはパレットコード通りに変化することになる．
5．作成された連続したデータから迷路を描く．…画面の表示

例，迷路サイズを 8×3 ブロックとした場合（1ブロックは座標データで 4×4 コマ）

3．座標データ

44	50	50	50	54	50	50	14
05	00	00	00	05	00	00	05
41	50	50	50	51	50	50	11

4．仮想迷路

00	00	00	00	00	00	00	00
00	FF	FF	FF	00	FF	FF	00
00	00	00	00	00	00	00	00

図5-5　迷路データから座標データを得る

成したら早速実行してみましょう．なお，本章の各プログラムも2,3章同様に次々とインクルードしていきます．また，ビープ音楽を利用しているので，リスト 4-1，リスト 4-2 もインクルードしてください．

このプログラムのなかで，CLPTXYという消去ルーチンがありますが，この内容は2，3章で使ったものと少し違っています．これは，このゲームのために必要になったもので，消去する色をパレット番号で指定できるようにしているのです．そして，その面を表示するか消去するかはキャリーフラグを利用して，00とFF$_H$を作り出すように工夫してあります．

また，座標データに関するルーチンは，図 5-5 を見ながらプログラムを追うと，わかりやすいと思います．迷路データを変えれば，どんな迷路でも表示してくれますので，テストしてみてください．ただし，このゲームでは主人公や敵の初期出現場所を，この迷路に合わせていますので，最後にはこのリストどおりのデータに戻しておかなければなりません．

リスト 5-1　迷路の表示(LIST 5-1.ASM)

```
;
;****** List 5-1-G ******
;
VTOP     equ      Ø                         VRAM のトップアドレス
HLEN     equ      8Ø                        1 ライン分の VRAM アドレスの増分
NEXTPT   equ      2ØØH                      NEXT  パターン用増分
BLUE     equ      ØA8ØØH                    VRAM の青色面のセグメント値
RED      equ      ØBØØØH                    VRAM の赤色面のセグメント値
GREEN    equ      ØB8ØØH                    VRAM の緑色面のセグメント値
;
GINIT:   ;Graphic system INITialize
         MOV      AX,4ØØØH                  グラフィック画面の表示開始コマンド番号をセット
         INT      18H                       BIOS コール
         MOV      AX,42ØØH                  グラフィック画面モード設定コマンド番号をセット
         MOV      CX,ØCØØØH                 640×400 ドット・カラー・モードで
         INT      18H                       ROM 内ルーチン・コール
         RET                                リターン
;
vwrite4  macro    vramseg                   マクロ定義スタート
         ifidn <vramseg>,<BLUE>             マクロパラメータが BLUE に等しいかチェック
           MOV      SI,BP                   等しければ SI ← BP
         else
           ADD      SI,DX                   等しくなければ SI ← SI+DX
         endif                              条件アセンブル・エンド
         MOV      AX,vramseg                AX にマクロパラメータを代入
```

```
        MOV     ES,AX                    AXを介してESにセグメント値セット
        MOV     DI,BX                    DIレジスタを初期化
        MOVSW                            ES:[DI]←DS:[SI]  &  DI←DI+2,SI←SI+2
        MOVSW                            ES:[DI]←DS:[SI]  &  DI←DI+2,SI←SI+2
        ifidn <vramseg>,<BLUE>           マクロパラメータがBLUEに等しいかチェック
          MOV    BP,SI                   等しければBP←SI
        endif                            条件アセンブル・エンド
        endm                             マクロ定義終了
;
DISP:   ;DISPlay
        PUSH    SI                       SIレジスタ値をスタックへ退避
        CALL    PDADR                    パターン番号から,データ・アドレスを求める
        CALL    XYADR                    表示アドレスを求めるため
        MOV     BX,DISAD                 表示アドレスをセット
        PUSH    DS                       データセグメント値をスタックへ退避
        MOV     AX,PTNSEG                パターン・データ用セグメント値をセット
        MOV     DS,AX                    データセグメントをセット
        MOV     CX,32                    Y方向ループ回数セット
        ADD     BP,4*32                  透明データ・カットとする
        MOV     DX,4*32-4                次のVRAMへの加算値
BXLP1:  ;BoX LooP 1
        vwrite4 BLUE                     パラメータをBLUEでマクロ展開
        vwrite4 RED                      パラメータをREDでマクロ展開
        vwrite4 GREEN                    パラメータをGREENでマクロ展開
        ADD     BX,HLEN                  次ラインの表示アドレス
        LOOP    BXLP1                    ループをCX回繰り返す
        POP     DS                       データ・セグメント復元
        POP     SI                       SIレジスタ復元
        RET                              リターン
;
CLPTXY: ;Clear PaTern  (X,Y)
        MOV     word ptr CLXY,BX         消去サイズ
        CALL    XYADR                    消去アドレスを求める
        MOV     AL,COLOR                 消去色の指定
        ROR     AL,1                     ALを右にローテート …… 青色面のビット情報をCYへ
        PUSH    AX                       AXレジスタ値をスタックへ退避
        SBB     AL,AL                    AL←AL-AL-CY(AL=0またはAL=FFHとなる)
        MOV     DX,BLUE                  B面のセグメント値をセット
        CALL    ERBOX                    DISPADからSIZEの大きさで四角形を描く
        POP     AX                       AXレジスタ値をスタックから復元
        ROR     AL,1                     ALを右にローテート …… 赤色面のビット情報をCYへ
        PUSH    AX                       AXレジスタ値をスタックへ退避
        SBB     AL,AL                    AL←AL-AL-CY(AL=0またはAL=FFHとなる)
        MOV     DX,RED                   B面のセグメント値をセット
        CALL    ERBOX                    DISPADからSIZEの大きさで四角形を描く
        POP     AX                       AXレジスタ値をスタックから復元
        ROR     AL,1                     ALを右にローテート …… ミドリ面のビット情報をCYへ
        SBB     AL,AL                    AL←AL-AL-CY(AL=0またはAL=FFHとなる)
        MOV     DX,GREEN                 G面のセグメント値をセット
        CALL    ERBOX                    DISPADからSIZEの大きさで四角形を描く
        RET                              リターン
```

```
;
ERBOX:   ;ERase BOX
         MOV      ES,DX                 DXレジスタを介してESへセグメント値を設定
         MOV      BP,DISAD              BP←消去アドレス
         MOV      DX,word ptr CS:CLXY   DX←消去のサイズ
         XOR      BX,BX                 BX←0
         XCHG     BL,DH                 BL←→DH
         SHL      DX,1                  DX←DX×2
ERL1:    ;ERase Loop 1
         MOV      DI,BP                 DIをBPで初期化
         MOV      CX,BX                 ストリング命令のループ回数セット
         REP      STOSB                 DS:[DI]←AL,DI←DI+1
         ADD      BP,HLEN               BPに次ラインのアドレスをセット
         DEC      DX                    DX←DX−1
         JNE      ERL1                  DXが0でなければER1へ
         RET                            リターン
;
CLXY     db       Ø,Ø                   SIZE
;
COLOR    db       Ø                     COLOR …… 消去色
;
VTOP     equ      Ø                     VRAMの先頭アドレス
;
vstosw   macro    vramseg               VRAMへのSTOSWマクロ定義
         MOV      CX,vramseg            CX←vramseg
         MOV      ES,CX                 CXレジスタを介してESへセグメント値セット
         MOV      CX,4Ø                 1ライン分のループ回数セット
         MOV      DI,BP                 VRAMアドレス初期化
         REP      STOSW                 ES:[DI]←AX
         endm                           マクロ定義終了
CLS:     ;CLear Screen                  ── 画面の高速消去
         MOV      BP,VTOP               BP←VRAMの先頭アドレス
         MOV      DX,4ØØ                DX←400ライン分のループ回数セット
         XOR      AX,AX                 AX←0
CLSLP1:  ;CLear Screen LooP 1
         vstosw   BLUE                  セグメント値BLUEでマクロ展開
         vstosw   RED                   セグメント値REDでマクロ展開
         vstosw   GREEN                 セグメント値GREENでマクロ展開
         MOV      BP,DI                 BP←DI
         DEC      DX                    DX←DX−1
         JNE      CLSLP1                DX=0でなければCLSLP1へ
         RET                            リターン
;
;****** List 5-1-N ******
;
XYADR:   ;XY  to  ADdress               ── (CL,CH)から表示アドレスを求める
         PUSH     AX                    AXレジスタ値をスタックへ退避
         XOR      AX,AX                 AXレジスタ初期化
         MOV      AH,CH                 Y座標の100H倍を求める
         SHR      AX,1                  Y座標の80H倍を求める
```

```
        SHL     CH,1                        Y座標の200H倍を求める
        ADD     AX,CX                       AX←Y×280H+X
        MOV     DISAD,AX                    表示アドレス保存
        POP     AX                          AXレジスタ復元
        RET                                 リターン

PDADR:  ;Pattern Data ADdRess               —— データ・アドレスをBPレジスタへ求める
        MOV     AH,Ø                        AXの上位アドレスを初期化
        SHL     AX,1                        AX←AX×2
        MOV     BP,offset PTNAD             BP←PTNADのオフセット・アドレス
        ADD     BP,AX                       データ・アドレス格納アドレスを求む
        MOV     BP,CS:[BP]                  パターン・データ・アドレスを求める
        RET                                 リターン

PTNAD   dw      NEXTPT*Ø                    PaTtern Data ADdress table
        dw      NEXTPT*1
        dw      NEXTPT*2
        dw      NEXTPT*3
        dw      NEXTPT*4
        dw      NEXTPT*5
        dw      NEXTPT*6
        dw      NEXTPT*7
        dw      NEXTPT*8
;
IMZX    equ     16                          Image MaZe X  迷路の横16ブロック
IMZY    equ     12                          Image MaZe Y  迷路の縦12ブロック
DOWN    equ     1                           下方向のラベル化
LEFT    equ     2                           左〃
RIGHT   equ     4                           右〃
UP      equ     8                           上〃
;
MZDATA  db      ØØH,ØØH,6AH,56H             MaZe Data ……迷路データ
        db      6AH,56H,ØAH,5ØH
        db      78H,1EH,Ø2H,4ØH
        db      5EH,7AH,1ØH,Ø8H
        db      75H,ØAEH,Ø5H,ØAØH
        db      6CH,36H,Ø1H,8ØH
;
IMAZE   db      252 dup (?)                 Image MAZE ……仮想迷路
;
AMAZE   db      192 dup (?)                 Arrow MAZE ……矢印迷路
;
MKIMZ:  ;Make Image MaZe                    —— 仮想迷路を作る
        MOV     BX,offset IMAZE             BX←仮想迷路の先頭アドレス
        MOV     DI,offset MZDATA            DI←迷路データの先頭アドレス
        MOV     CX,IMZX+2                   CX←迷路の横サイズ+2
MILP1:  ;Make Image maze LooP 1
        MOV     byte ptr [BX],ØFFH          迷路の横サイズ+2だけFFHを入れる
        INC     BX
        LOOP    MILP1
        MOV     CX,IMZY                     CX←迷路の縦サイズ
```

```
MILP2:  ;Make Image maze LooP 2
        MOV     byte ptr [BX],ØFFH      仮想迷路左端に FFH を入れる
        INC     BX
        MOV     DX,2                    横のサイズ←8ブロック×2
MILP3:  ;Make Image maze LooP 3
        PUSH    CX                      CX の値をスタックへ退避
        MOV     AL,[DI]                 迷路データ
        INC     DI                      DI レジスタ更新
        MOV     CX,8                    ループ回数セット
MILP4:  ;Make Image maze LooP 4
        MOV     AH,Ø                    ┌ AL をローテートし,CY で1,0 を判断
        ROL     AL,1                    │ 1のとき …… AH=FFH
        JNB     MILP5                   └ 0のとき …… AH=0
        DEC     AH
MILP5:  ;Make Image maze LooP 5
        MOV     [BX],AH                 仮想迷路データ
        INC     BX
        LOOP    MILP4                   迷路データ全ビット, CX 回 MILP4 を繰り返す
        POP     CX                      スタックから CX レジスタ値を復元
        DEC     DX                      DX←DX-1
        JNE     MILP3                   DX 回 MILP3 を繰り返す
        MOV     byte ptr [BX],ØFFH      仮想迷路の右端に FFH を入れる
        INC     BX                      仮想迷路のアドレスを次の段にする
        LOOP    MILP2                   迷路の縦サイズだけ MILP2 を繰り返す
        MOV     CX,IMZX+2
MILP6:  ;Make Image maze LooP 6         ┐
        MOV     byte ptr [BX],ØFFH      │ 仮想迷路の最下段
        INC     BX                      │ 迷路の横サイズ+2 だけ FFH を入れる
        LOOP    MILP6                   ┘
        RET                             リターン
;
MKAMZ:  ;MaKe Arrow MaZe                ── 矢印迷路を作る
        MOV     BX,offset AMAZE         BX←矢印迷路の先頭アドレス
        MOV     DI,offset IMAZE         DI←実際に迷路の始まる仮想迷路アドレス
        ADD     DI,IMZX+3
        MOV     CX,IMZY                 CX←迷路の縦サイズ
MALP1:  ;MaKe Arrow maze Loop 1
        MOV     DX,IMZX                 DX←迷路の横サイズ
MALP2:  ;MaKe Arrow maze Loop 2
        MOV     byte ptr [BX],Ø         まだ矢印は立たない
        MOV     AL,[DI]                 仮想迷路データ
        OR      AL,AL                   AL=0 か?
        JE      NNPOS                   AL=0 であれば NNPOS へ
        JMP     NPOS                    NPOS へジャンプ
NNPOS:  ;Not Next POSition
        DEC     DI
        MOV     AL,[DI]                 AL←左側の仮想迷路データ
        INC     DI                      DI レジスタ更新
        OR      AL,AL                   AL=0 か?
        JNE     ARCKR                   AL=0 でなければ ARCKR へ
        MOV     AL,LEFT                 AL←左矢印
```

```
        ADD     [BX],AL                 [BX]←[BX]+AL
ARCKR:  ;ARrow CheckK Right
        INC     DI                      DIレジスタ更新
        MOV     AL,[DI]                 AL←右側の仮想迷路データ
        OR      AL,AL                   AL=0か？
        JNE     ARCKU                   AL=0でなければARCKUへ
        MOV     AL,RIGHT                AL←右矢印
        ADD     [BX],AL                 [BX]←[BX]+AL
ARCKU:  ;ARrow CheckK Up
        ADD     DI,-IMZX-3              DI←DI−迷路の横サイズ−3
        MOV     AL,[DI]                 ALは上側の仮想迷路データ
        OR      AL,AL                   AL=0か？
        JNE     ARCKD                   AL=0でなければARCKD
        MOV     AL,UP                   AL←上矢印
        ADD     [BX],AL                 [BX]←[BX]+AL
ARCKD:  ;ARrow CheckK Down
        ADD     DI,IMZX+IMZX+4          DI←DI+迷路の横サイズ×2+4
        MOV     AL,[DI]                 ALは下側の仮想迷路データ
        OR      AL,AL                   AL=0か？
        JNE     MAPOS                   AL=0でなければMAPOSへ
        MOV     AL,DOWN                 AL←下矢印
        ADD     [BX],AL                 [BX]←[BX]+AL
MAPOS:  ;Make Arrow POSition
        ADD     DI,-IMZX-2              DI←DI−迷路の横サイズ−2
NPOS:   ;Next POSition
        INC     DI                      仮想迷路アドレス更新
        INC     BX                      矢印迷路アドレス更新
        DEC     DX                      DX←DX−1
        JE      NMAPL2                  DX=0であればNMAPL2へ
        JMP     MALP2                   DX回MALP2を繰り返す
NMAPL2: ;Not MaKe Arrow maze Loop 2
        ADD     DI,2                    DI←DI+2：仮想迷路
        LOOP    MALP1                   CX回MALP1を繰り返す
        RET                             リターン
;
RMEDGE: ;ReMove EDGE                    ── 仮想迷路から周囲のFFHをとる
        MOV     BX,CS                   BX←CS
        MOV     ES,BX                   BXレジスタを介してCSレジスタ値を格納
        MOV     DI,offset IMAZE         DI←仮想迷路の先頭アドレス
        MOV     SI,DI                   SI←DI
        ADD     SI,IMZX+3               SI←実際に迷路の始まる仮想迷路アドレス
        MOV     AX,12                   迷路の縦ブロック数
RELOOP: ;Remove Edge LOOP
        MOV     CX,8                    迷路の横ブロック数÷2
        REP     MOVSW                   ES：[DI]←DS：[SI]をCX回繰り返す
        ADD     SI,2                    SI←SI+2
        DEC     AX                      AX←AX−1
        JNE     RELOOP                  迷路の縦ブロック数だけRELOOPを繰り返す
        RET                             リターン
```

```
;
WALPT   equ     Ø                       WALl PaTtern number
;
DISPMZ: ;DISPlay MaZe                   ―― 迷路の表示
        MOV     BX,offset IMAZE         BX←仮想迷路の先頭アドレス
        MOV     CX,ØØØØH                CX←0
DMLOOP: ;Display Maze LOOP
        PUSH    CX                      CXの値をスタックへ退避
        PUSH    BX                      BXの値をスタックへ退避
        MOV     AL,[BX]
        CMP     AL,ØFFH                 AL=FFH(壁)ならDPWALLへ
        JE      DPWALL
        MOV     [COLOR],AL              カラー番号=ALでそのマスを消去する
        MOV     BX,41ØH                 BOXサイズを指定
        CALL    CLPTXY                  BXでBOX型を描く
        JMP     SEEKNP                  SEEKNPへジャンプ
DPWALL: ;DisPlay WALL
        MOV     AL,WALPT                AL←壁のパターン番号セット
        CALL    DISP                    (CL,CH)にAL(壁)を表示
SEEKNP: ;SEEK Next Position
        POP     BX                      BXの値をスタックから復元
        POP     CX                      CXの値をスタックから復元
        INC     BX                      BX←BX+1 …… 仮想迷路アドレス更新
        ADD     CL,4                    CL←CL+4 …… X座標を次のブロックにする
        MOV     AL,CL                   AL←CL
        CMP     AL,61                   CL<61ならDMLOOPへ
        JNB     NDMLP1
        JMP     DMLOOP
NDMLP1: MOV     CL,Ø                    CL←0
        ADD     CH,4                    CH←CH+4 …… 次段のY座標
        MOV     AL,CH                   AL←CH
        CMP     AL,45                   CH<45ならDMLOOPへ
        JNB     NDMLP2
        JMP     DMLOOP
NDMLP2: RET                             リターン
;
DATLD:  ;DATa LoaD
        MOV     SI,offset LODTBL        SI←ロード・テーブル・アドレス
DLDLP1: ;Data LoaD LooP 1
        MOV     AL,[SI].CMD             AL←ロードコマンド
        AND     AL,AL                   コマンドエンドか?
        JE      DLD2                    コマンドエンドであればDLD2へ
        CALL    LOAD                    データ・ロード
        ADD     SI,type lodinf          SIレジスタ更新
        JMP     DLDLP1                  DLDLP1へ
DLD2:   ;Data LoaD 2
        RET
```

```
;
LOAD:    ;LOAD
         LEA     DX,[SI].PASS          ファイルパス名格納アドレス取得
         MOV     AL,Ø                  ファイル・アクセス・コントロール
         MOV     AH,3DH                ハンドルのオープン
         INT     21H                   ファンクションコール
         JNB     DOPEN                 エラーがなければDOPENへ
         CMP     AX,2                  ファイルが存在しない場合のチェック
         JNE     DLERR                 エラー・コードによる処理の選択
         LEA     DX,[SI].PASS          ファイルパス名格納アドレス取得
         MOV     AH,9                  ストリングのスクリーン出力
         MOV     [SI].ENDSIN,"$"       ストリング終了コードセット
         INT     21H                   ファンクションコール
         print   EMES2                 エラー・メッセージ出力
         MOV     AX,ØFFFFH             エラー・サインをセット
         JMP     DLRET                 リターン
DLERR:   ;Data Load ERRor
         LEA     DX,[SI].PASS          ファイルパス名格納アドレス取得
         MOV     AH,9                  ストリングのスクリーン出力
         MOV     [SI].ENDSIN,"$"       ストリング終了コードセット
         INT     21H                   ファンクションコール
         print   EMES1                 エラー・メッセージ出力
         MOV     AX,ØFFFFH             エラー・サイン・セット
         JMP     DLRET                 リターン
DOPEN:   ;Data file OPEN
         MOV     HANDL,AX              ハンドル保存
         MOV     BX,AX                 ハンドルを指定レジスタへ
         MOV     CX,Ø                  CX←0
         MOV     DX,CX                 DX←0
         MOV     AH,42H                ファイル・ポインタの移動とする
         MOV     AL,2                  ポインタをファイルの終わりに移動とする
         INT     21H                   ファイルの大きさをAXレジスタへ
         MOV     FLLEN,AX              ファイルの大きさを保存
         MOV     BX,HANDL              ハンドルを指定レジスタへ
         MOV     AH,3EH                ハンドルのクローズ
         INT     21H                   ファンクションコール
         LEA     DX,[SI].PASS          指定ファイルのパス名の格納アドレス取得
         MOV     AL,Ø                  ファイル・アクセス・コントロール
         MOV     AH,3DH                ハンドルのオープン
         INT     21H                   ファンクションコール
         MOV     HANDL,AX              ハンドルの保存
         MOV     BX,AX                 指定レジスタにハンドルセット
         PUSH    DS                    データ・セグメント値をスタックへ退避
         MOV     DX,[SI].LDADR         ロード・アドレスをセット
         MOV     CX,FLLEN              ファイルの大きさをセット
         MOV     AX,[SI].LDSEG         ロード・セグメント値セット
         MOV     DS,AX                 DSへAXを介してセグメント値セット
         MOV     AH,3FH                データ・ロード
         INT     21H                   ファンクションコール
         POP     DS                    DSをスタックから復元
         MOV     FILEL,AX              読み込んだバイト数保存
```

```
        MOV     BX,HANDL                                  ハンドルを指定レジスタへ
        MOV     AH,3EH                                    ハンドルのクローズ
        INT     21H                                       ファンクションコール
        XOR     AX,AX                                     ノーマル・リターンとする
DLRET:  ;Data Load RET
        RET                                               リターン
;
FILEL   dw      Ø                                       ┐
FLLEN   dw      Ø                                       │
DISAD   dw      Ø                                       ├ ファイル・アクセス用ワークエリア
HANDL   dw      Ø                                       ┘
EMES1   db      1Ø,13                                     エラー・メッセージ番号1
        db      "ファイル オープン エラー"
        db      1Ø,13,"$"
EMES2   db      1Ø,13                                     エラー・メッセージ番号2
        db      "ファイル ガ, アリマセン "
        db      1Ø,13,"$"
CLEAR   db      1BH,"[2J",1BH,"[>1h",1BH,"[>5h$"
CSRON   db      1BH,"[>51",1BH,"[>11$"

CODE    ends                                              命令の置かれているセグメントの終わり

STSEG   segment STACK                                     スタック用セグメントの開始
        db      1ØØH dup (?)                              データ域を100Hバイト確保
STSEG   ends                                              スタック用セグメントの終わり

PTNSEG  segment                                           パターン用セグメントの開始
        db      8ØØØH dup (ØFFH)                          データ域を8000Hバイト確保
PTNSEG  ends                                              パターン用セグメントの終わり

        end                                               プログラム・エンド
```

テスト5-1　テスト・プログラム(TEST 5-1.ASM)

```
;****** TEST 5-1 ******
CODE    segment                               命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,SS:STSEG

print   macro   string                        文字列出力マクロの定義
        LEA     DX,string                     マクロパラメータのオフセットを DX へ
        MOV     AH,9                          ファンクションコール番号 9 …… 文字列出力
        INT     21H                           ファンクションコール
        endm                                  マクロ定義終了
;
PTEST:  ;Plogram TEST
        CLD                                   ディレクション・フラグ・リセット
        MOV     AX,CS                         AX ← CS
        MOV     DS,AX                         AX レジスタを介して DS に CS を格納
        print   CLEAR                         テキスト画面クリア
        CALL    DATLD                         パターン・データ・ロード
        CALL    GINIT                         グラフィック・システム初期化
        CALL    CLS                           画面クリア
        CALL    MKIMZ                         仮想迷路を作る(周囲は FFH)
        CALL    MKAMZ                         矢印迷路を作る
        CALL    RMEDGE                        仮想迷路から周囲の FFH を取る
        CALL    DISPMZ                        迷路を表示
        print   CSRON                         カーソル・オン
        MOV     AL,Ø                          リターン・コード・ノーマル
TEXIT:
        MOV     AH,Ø4CH                       プロセスの終了
        INT     21H                           ファンクションコール
;
lodinf  struc                                 ロード情報構造体定義
CMD     db      Ø                             ロード・コマンド
PASS    db      "           "                 パス格納領域 (11 文字分)
ENDSIN  db      Ø                             パス・エンド・コード
LDADR   dw      Ø                             ロード・アドレス
LDSEG   dw      Ø                             ロード・セグメント
lodinf  ends                                  構造体定義終了
;
LODTBL  lodinf <1,"MEIRO.DAT"  ,Ø,Ø,PTNSEG>
        lodinf < >
;
        include LIST5-1.ASM                   LIST5-1.ASM を取り込む
```

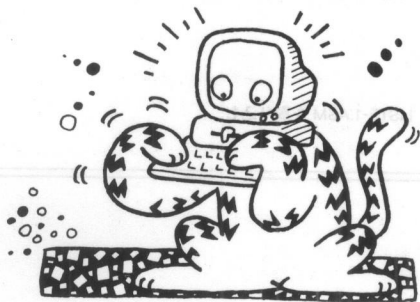
3. キー入力… 操作性の向上

本章のVIP(?)である迷路については，すでにその全貌が明らかになっています．パターンの移動に必要な座標データや，迷路を画面に表示するルーチンも実際に作ってしまったわけですから，ここから先は付け足しに過ぎないかもしれません．別の言い方をすれば，本節は迷路ゲームを完成させるためにあるようなものです．しかし，1つのゲームを作るにあたって，めんどうな作業が多くなるのもこれからです．とくに，現実に商品となる作品を作る際には，ここからどの程度ゲームに対して「気配り」をできるかがポイントとなってきます．

いわゆる原因不明のバグが出てくるのも，これから先にかけてがほとんどですし，単純に迷路についてわかったからといって，気を抜くことはまだまだできません．

ここで作るゲームは，マシン語勉強用であって商品ではありませんから，すべてにわたって「気配り」をすることはしませんが，基本的な部分においては細かな点にまで気を遣っています．では，このような迷路型ゲームにおける基本的な部分とは何かというと，それはキー操作が簡単かどうかということです．どんなにアイデアがよくても，主人公を思いどおりに動かせなければ，ゲームがつまらなくなってしまいます．つまり，キー操作もゲームのアイデアの重要な要素である，ということなのです．ただし，キー操作だけすぐれていても，ゲームとしての価値はありませんから，やはりメインのアイデアが最重要であることに変わりはありません．ですから，ここでのキー操作法が，すべての迷路型ゲームについて最適であるとは断言できませんし，そういうものはまた存在しないものなのです．そのため，オリジナルのゲームを作る際には，そのゲーム内容に応じたキー操作のアイディアも一緒に考える必要があります．

さて，本題にはいる前にこの迷路ゲームで使うパターンを一気に作成してしまいましょう．(パターンは巻頭口絵4)．本当は，迷路では方向別に2～3パターンを用意すると，アニメ的に動きが出て良いのですが，テストということで方向別に作るのはやめて，それぞれ2パターンを交互に表示するだけで妥協しました．もし，プログラム・コンテストに応募しようと思っている方は，すべてにおいて安易に妥協などしてはいけません．だれが見ても，妥協した所はすぐにわかるものです．どうせなら，これは作るのがたいへんだったろうな，と想像されるようにしておくべきです!?

ところで，アニメーションといえば，ウォ

ルト・ディズニーのものがキャラクタといい，動きのなめらかさといい，正に世界の最高峰ですが，彼は決して妥協をしなかったそうです．それどころか，つねに新しいものへのチャレンジを試み，かならず前作より優れた作品を生み出していったのです．しかし，その陰に隠れて，意外と知られていないのがアブ・アイワークスの存在です．彼の絵を描く能力，機械を扱う能力というものは天才的であり，1日に700枚ものデッサンを描いたという逸話があるくらいです．ミッキー・マウスの生みの親がディズニーなら，育ての親とも生命を吹き込んだ男とも言われているのが，アブ・アイワークスなのです．ディズニーの夢と希望を持つ雰囲気を，PC-9801のゲームで見たいと思うのですが，残念ながらまだそのような作品に出会ったこともまた作ろうとしても足元にも及ばないというのが現実です．どこかに和製アブ・アイワークスはいませんかね……．さて，パターン・エディタで作成したパターン・データは，本書ではMEIRO.DATというファイルとしてロードするようになっていますが，このファイル名は適当に設定してください．

また，数字，文字のデータは前回と同じものですが，アドレスが違っていますので，同様に転送をする必要があります．

パターン・データが整えば，残るはプログラムということになります．が，その前にキー・スキャンから移動方向を決定するまでのアルゴリズムを，正確に把握しておかないと，このプログラムは少々難解です．

それは，移動方向の決定に際し，座標データによる制限が加わっているだけでなく，

■ 移動カウンターが0→座標データが存在する

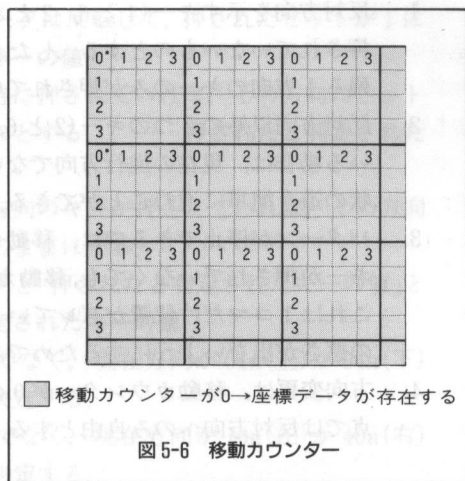



図 5-6　移動カウンター

キー操作を簡単にするというテーマもはいっているからです．方向番号は前の節で決めたとおりですが，追加として停止状態を方向番号＝0としています．

また，反対方向以外への方向変更は4コマごとにしかできないということを判断するため，図 5-6 のように座標データのある位置を基準(0)として，方向別に移動カウンター(0～3)を定めています．この値は，ワークエリア(MYWORK＋1)上に保存しておき，移動するたびに方向により増減されることになります．

さてキー入力に対する操作性アップのためのルールは次のページに示してあります．迷路ゲームではごく当り前のものです．

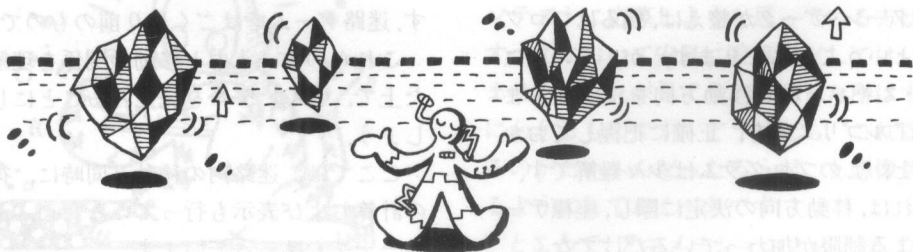
これらのキー入力と移動の関係を理解した上で，リスト 5-4 を見てみることにしましょう．

ここでは，迷路内の移動と同時に，得点の計算および表示も行っているため，そのルーチンもはいっています．

1. 反対方向を示すキー(4と6，2と8)が，同時に押されている場合は，両方共に押されていないものとする．したがって，3つのキーが同時に押されたときは，残る1方向のキーのみが押されていることになる．
2. 反対方向以外の2つのキー(2と6，6と8，8と4，4と2)が，同時に押されている場合は，現在の進行方向でない方向に優先権がある．これにより，カギ型状の道を簡単に進むことができる．
3. パターンが停止できるのは，移動カウンターが0になっている場所だけとし，キーが押されていなくても，移動カウンターが0になるまでは同じ方向に進む．これは1コマだけ位置がズレているために曲れない，というようなキー操作性の悪さが出ないようにするためである．
4. 方向変更は，移動カウンターが0の所では，座標データにより，それ以外の地点では反対方向へのみ自由とする．

得点は道路を塗るたびに，パレット番号に相当するだけのアップがあります．また，ボーナスとして，全部の道路が同一色になったとき(白は除く)には，そのパレットコード×1000点が加算されます．ただし，道路(ブロック)の数を数えるルーチンがまだありませんから，現在はボーナス点は正しく加算されません．同様に，スコアについても初期値(0点)を入れていませんから，初期の得点は不定です．これらの得点に関しては，プログラム的に前回のものと変わりがありません．このプログラムで中心となるのは，やはり主人公の移動(MYMOVE)についてのルーチンで，なかでも方向を決定している部分(KEYCHK)がキーポイントです．

方向が決まれば，パターンの下にある道路の色を仮想画面(IMAZE)から取り出し，方向別に消去してパターンを移動させればいいのです．移動カウンターの増減や，得点の加算なども方向が決定してからのことになります．

その方向決定までのプログラムの手順ですが，わかりやすく書くと次のようになります．

1. 押されたキー(2, 4, 6, 8)を調べる．データは反転して，押されたビットが1になるようにしておく．　……**押されたキーの値**
2. 反対方向のキー(2と8, 4と6)が，同時に押されていれば，その方向のビットを両方共0にして，押されていないものとする．　……**押されたと判定されたキーの値**
3. 移動カウンターが0でない場合，反対方向のキーが押されていれば，その方向に決定し，それ以外はすべて現在方向のままにする．
4. 移動カウンターが0の場合，座標データと「押されたと判定されたキーの値」とのANDを取る．　……**移動可能と判定されたキーの値**
5. 「移動可能と判定されたキーの値」が0でなく，現在方向が01H(上)か04H(下)のときは，左右への方向から優先的に決定する．
6. 「移動可能と判定されたキーの値」が0でなく，現在方向が10H(左)か40H(右)のときは，上下への方向から優先的に決定する．
7. 「移動可能と判定されたキーの値」が0の場合は，停止(方向番号=0)とする．

なお，今までキーのチェックには簡単に利用できるROM内ルーチンを利用してきました．しかし，簡単であるというメリットの陰に，実は困った問題が残されていたのです．というのは，リアルタイム·ゲームで使う処理としてはスピードが遅いということです．

そこで，本章のプログラムからは，キーボードが押されたり離されたりしたときにかかる，キーボードからの割り込みを直接利用することにします．

CPUは，何らかの割り込みがかかると，それまで実行していた仕事をやめて，あらかじめ用意してあるジャンプ·テーブルから，割り込みの原因にそったエントリを参照し，目的の割り込み処理ルーチンへと制御を移します．

表5-1を見るとわかるように，割り込みテーブルのベクタ番号9がキーボードに対するベクタですから，キーボード割り込みを利用したい場合には，ここをゲーム専用のエントリに書き換えることになります．

ベクタ番号	用　途
0	除算エラー
1	シングル・ステップ
2	NMI
3	INT3
4	オーバーフロー
5	ハードコピー(COPY)キー
6	STOPキー
7	インターバル・タイマー
8	タイマー
9	キーボード
10	CRTV(VSYNC)
⋮	⋮

表5-1　割り込みテーブル

MS-DOSにはこれらの割り込み処理を管理するためのファンクションがありますので，これを利用すると簡単にエントリを書き換えることができます．

さて，専用のルーチンへのジャンプが可能になれば，問題となるのはキーボードからの情報をいかに取得するかですが，この場合はさきほどの割り込みテーブルのエントリの書き換えのように簡単にはいきません．

キーボードとの情報のやりとりはポート41H，43Hを介して行われます(**表5-2**)．41Hはキーデータ取得用で，43Hがコマンドを送ったり状態のチェックなどに使われるものです(**表5-3**，**表5-4**，**表5-5**)．

キー情報の入力の手順としては，ポートのステータスを読み，エラーをチェックし，正常であればデータを入力して，さらにそのデータを加工するという作業をすることになります．

ポート41Hからの入力データは**表5-6**を参照してください．

たとえば，スペースキーが押されたときには，表5-6から34H，離されたときはB4Hとなります．リスト5-4では使うと思われるキーデータを情報として取り込み，専用の領域を用意して，プログラムで使いやすいデータに加工してから保存しています．このリストでKEYSETというルーチンが実際の割り込み処理をしている部分です．

割り込みというのは，実際にいつかかるかわかりませんから，割り込み処理ルーチンでは使うレジスタを退避しておかないと，もとのプログラムに復帰したときに正常に動作しなくなってしまいます．ここでは，初めに退避して処理の最後でもとに戻しています．また，割り込み処理の終わりには割り込みコントローラに対して，割り込み処理が終ったことを知らせるためにEOI(End Of Interrupt)を発行しています．

割り込み処理からリターンするときにはIRET命令を使っていることに注意してください．これは，割り込み処理に制御が移るときに，オフセットアドレスとセグメントアドレス，さらにフラグがPUSHされるからです．したがって，これらをPOPして処理を返す命令IRETが使われるのです．

キーボード割り込みのエントリの登録は，MS-DOSのファンクションを利用して行っています．SETINITが登録用のルーチンで，テストプログラムの先頭でコールします．また，テストプログラムの終わりではORIINTでキーボード割り込みを初期状態に戻しています．この作業を忘れてしまうと，キーボードからの入力はまったく受け付けてくれませんのでくれぐれも忘れないようにしてください．

では，テストを実行してプログラムの動作を確認してみましょう．ウェイトを入れていないので，動きが速いかもしれませんが，キーの操作性は合格点のはずです．ゲームのアイデアに対する点数は，……これは，好みの問題(？)ですから，あなた自身の判断で採点をお願いいたします．といっても，まだゲームとして完成していないのですから，無理な話でしたね．先へ進みましょう……．

I/O・コントロール		b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
モード・ライト	43H	S_2	S_1	P	PEN	L_2	L_1	B_2	B_1
コマンド・ライト	43H	*	IR	KB	ER	RST	RX	RTY	TX
ステータス・リード	43H	*	*	FE	OE	PE	*	RDY	*
データ・リード	41H	表5-6参照							

表5-2　キーボードのI/Oコントロール

モード・ライト	解　説			
B_2B_1：ボーレート	00：同期モード	01：*1モード	10：*16モード	11：*64モード
L_2L_1：キャラクタ長	00：5ビット	01：6ビット	10：7ビット	11：8ビット
PEN：パリティ許可	0：ディスエーブル	1：イネーブル		
P：パリティ	0：奇数	1：偶数		
S2S1：ストップビット	00：無効	01：1ビット	10：1.5ビット	11：2ビット

表5-3　モード・ライト

コマンド・ライト	解　説	
TX：送信	0：ディスエーブル	1：イネーブル
RTY：リトライ	0：イネーブル	1：ディスエーブル
RX：受信	0：ディスエーブル	1：イネーブル
RST：リセット	0：ディスエーブル	1：イネーブル
ER：エラーリセット	0：	1：エラーフラグ・リセット
KB：KB送信	0：イネーブル	1：ディスエーブル
IR：内部リセット	0：	1：モードライト命令受付状態に戻す

表5-4　コマンド・ライト

ステータス・リード	解　説	
RDY	インターフェイス信号RDYと同じ	
PE：パリティエラー	0：	1：パリティ・エラー検出
OE：オーバーランエラー	0：	1：オーバーランエラー検出
FE：	0：	1：ストップビット未検出

表5-5　ステータス・リード

押した時の出力データ

D7	D6	D5	D4									
					0	0	0	0	0	0	0	0
					0	0	0	0	1	1	1	1
					0	0	1	1	0	0	1	1
					0	1	0	1	0	1	0	1
D3	D2	D1	D0		0	1	2	3	4	5	6	7
0	0	0	0	0	ESC	Q タ	F ハ	, < 、 ネ	ー	・	STOP	SHIFT
0	0	0	1	1	1 ! ヌ	W テ	G キ	. > 。 ル	／	NFER	COPY	CAPS
0	0	1	0	2	2 " フ	E ィ イ	H ク	/ ? ・ メ	7		f・1	カナ
0	0	1	1	3	3 # ァ ア	R ス	J マ	_ ロ	8		f・2	GRPH
0	1	0	0	4	4 $ ゥ ウ	T カ	K ノ	SPACE	9		f・3	CTRL
0	1	0	1	5	5 % ェ エ	Y ン	L リ	XFER	*		f・4	
0	1	1	0	6	6 & ォ オ	U ナ	; + レ	ROLL UP	4		f・5	
0	1	1	1	7	7 ' ャ ヤ	I ニ	: * ケ	ROLL DOWN	5		f・6	
1	0	0	0	8	8 (ュ ユ	0 ラ	] } 」 ム	INS	6		f・7	
1	0	0	1	9	9) ョ ヨ	P セ	Z ッ ツ	DEL	+		f・8	
1	0	1	0	A	0 ヲ ワ	@ ~ ゛	X ' サ	↑	1		f・9	
1	0	1	1	B	- = ホ	[{ 「 ・	C ソ	←	2		f・10	
1	1	0	0	C	^ へ	↵	V ヒ	→	3			
1	1	0	1	D	¥ ¦ ー	A チ	B コ	↓	=			
1	1	1	0	E	BS	S ト	N ミ	HOME CLR	0			
1	1	1	1	F	TAB	D シ	M モ	HELP	,			
D7	D6	D5	D4	D3 D2 D1 D0	8	9	A	B	C	D	E	F
					0	1	0	1	0	1	0	1
					0	0	1	1	0	0	1	1
					0	0	0	0	1	1	1	1
					1	1	1	1	1	1	1	1

離した時の出力データ

表5-6 キーコード表

リスト5-2 テンキーによる移動(LIST 5-2.ASM)

```
;
;****** List 5-2 ******
;
PRETKEY equ 1CH
PHCLR   equ ØBEH
PESC    equ ØØH
PSPACE  equ 34H
KSPACE  equ ØB4H
PDOWN   equ 4BH
KDOWN   equ ØCBH
PLEFT   equ 46H
KLEFT   equ ØC6H
PRIGHT  equ 48H
KRIGHT  equ ØC8H
PUP     equ 43H
KUP     equ ØC3H
;
KEYSET: ;KEY SET
        PUSH    AX                      ┐
        PUSH    BX                      │
        PUSH    CX                      │ 使用レジスタの退避
        PUSH    DX                      │
        PUSH    DI                      │
        PUSH    ES                      ┘
        IN      AL,43H                  AL←キー・ステータス
        TEST    AL,38H                  エラーチェック
        JNZ     KEYOUT                  エラーがあればKEYOUTへ
        MOV     AL,16H                  コマンド・セット
        OUT     43H,AL                  コマンド送出
        IN      AL,41H                  AL←キー・データ・リード
        CMP     AL,PESC                 キーデータはESCが押されたか？
        JNE     NSTOP                   押されていなければNSTOPへ
        MOV     byte ptr CS:KPESC,1     STOPサインをオン
        JMP     KEYOUT                  KEYOUTへ
NSTOP:  ;Not STOP key
        CMP     AL,PRETKEY              キーデータはRETURNが押されたか？
        JNE     NRETK                   押されていなければNRETKへ
        MOV     byte ptr CS:RETKEY,1    RETURNサインをオン
        JMP     KEYOUT                  KEYOUTへ
NRETK:  ;Not RETurn key
        MOV     BX,CS                   BX←CS
        MOV     ES,BX                   BXレジスタを介してESへCSレジスタ値を格納
        MOV     DI,offset KEYTBL        DI←キー情報テーブル先頭アドレス
        MOV     CX,1Ø                   ストリング命令用ループ回数セット
        REPNZ   SCASB                   キーボード割り込みの要因チェック
        JNZ     KEYOUT                  目的の要因でないときはKEYOUTへ
        MOV     DI,offset KEYINF+9      要因別キーデータ・テーブルのエンド・アドレス・セット
        SUB     DI,CX                   キーデータ・テーブル・アドレスの補正
        TEST    AL,8ØH                  キーが押されたか否かのチェック
        JNE     KEYOF                   キーが離されたのであればKEYOFへ
```

```
        MOV     AL,CS:[DI]              押されたキーに対応する情報を取り出す
        OR      CS:KEYDAT,AL            押されたキーに対するビットをセットする
        JMP     KEYOUT                  KEYOUTへジャンプ
KEYOF:  ;KEY OFf
        MOV     AL,CS:[DI]              離されたキーに対応する情報を取り出す
        AND     CS:KEYDAT,AL            離されたキーに対するビットをクリアする
KEYOUT: ;KEY OUT
        MOV     AL,20H                  EOIコードセット
        OUT     0,AL                    EOIの発行
        POP     ES
        POP     DI
        POP     DX                      使用レジスタの復元
        POP     CX
        POP     BX
        POP     AX
        IRET                            リターン
;
KEYDAT  db      0                       キーデータ保存用
;
KEYTBL  db      PSPACE,PDOWN            要因別キー情報テーブル
        db      PLEFT,PRIGHT
        db      PUP,KSPACE
        db      KDOWN,KLEFT
        db      KRIGHT,KUP
;
KEYINF  db      80H,1H,2H,4H            要因別データ・テーブル
        db      8H,07FH,0FEH
        db      0FDH,0FBH,0F7H
;
KPESC   db      0                       STOPキー用領域
;
KPKOFF  dw      0                       キーボード割り込みオフセット・アドレス保存用
;
KPKSEG  dw      0                       キーボード割り込みセグメント・アドレス保存用
;
RETKEY  db      0                       RETURNキー用領域
;
IREDKEY equ     3509H                   キーボード割り込みベクタを求めるため
;
ISETKEY equ     2509H                   キーボード・ベクタをセットするため
;
SETINT: ;SET INTerrupt
        MOV     AX,IREDKEY              キーボード割り込みベクタを得るため
        INT     21H                     ファンクションコール
        MOV     KPKOFF,BX               キーボード割り込みオフセットアドレス保存
        MOV     KPKSEG,ES               キーボード割り込みセグメントアドレス保存
        MOV     AX,ISETKEY              専用の割り込みベクタ・セット用
        MOV     DX,offset KEYSET        DXへ専用のルーチンのオフセット・アドレスを格納
        INT     21H                     割り込みベクタ・セット
        OUT     64H,AL                  インターラプトリセット：ダミーをOUTする
        RET                             リターン
```

```
;
ORIINT: ;ORIginal INTerrupt set
        PUSH    DS                  DSレジスタ値をスタックへ保存
        MOV     DS,CS:KPKSEG        もとのセグメントアドレス値をセット
        MOV     DX,CS:KPKOFF        もとのオフセットアドレス値をセット
        MOV     AX,ISETKEY          キーボード割り込みベクタ・セット用
        INT     21H                 ファンクションコール
        POP     DS                  DSレジスタ値をスタックから復元
        RET                         リターン

DISPLE: ;DISPlay LEtter             ── 文字・数字の表示
        PUSH    SI                  SIレジスタ値をスタックへ退避
        PUSH    DS                  DSレジスタ値をスタックへ退避
        CALL    XYADR               表示アドレスを求めるため
        CALL    SEEKLD              データのアドレスを求めるため
        MOV     SI,AX               SI, AX
BOXL:   ;BOX of Letter              ── 1文字の表示
        MOV     DI,DISAD            DI←表示アドレス
        MOV     CX,16               CX←縦のドット数
        MOV     AX,PTNSEG           文字列パターンの格納されたセグメント・アドレス
        MOV     DS,AX               AXレジスタを介してDSにセグメント値を格納
LLOOP:  ;Letter LOOP
        MOV     AX,BLUE             AX←B面セグメント値
        MOV     ES,AX               AXを介してESにセグメント値を格納
        MOV     DX,[SI]             DX←DS:[SI]
        AND     ES:[DI],DX          ES:[DI]←ES:[DI] AND DX
        MOV     AX,[SI+20H]         AX←DS:[SI+20H]
        OR      ES:[DI],AX          ES:[DI]←ES:[DI] OR AX
        MOV     AX,RED              AX←R面セグメント値
        MOV     ES,AX               AXを介してESにセグメント値を格納
        AND     ES:[DI],DX          ES:[DI]←ES:[DI] AND DX
        MOV     AX,[SI+40H]         AX←DS:[SI+40H]
        OR      ES:[DI],AX          ES:[DI]←ES:[DI] OR AX
        MOV     AX,GREEN            AX←G面セグメント値
        MOV     ES,AX               AXを介してESにセグメント値を格納
        AND     ES:[DI],DX          ES:[DI]←ES:[DI] AND DX
        MOV     AX,[SI+60H]         AX←DS:[SI+60H]
        OR      ES:[DI],AX          ES:[DI]←ES:[DI] OR AX
        ADD     DI,HLEN             表示アドレスを次ラインにする
        ADD     SI,2                SI←SI+2
        LOOP    LLOOP               CX回ループ
        POP     DS                  スタックからDSレジスタ値を復元
        POP     SI                  スタックからSIレジスタ値を復元
        RET                         リターン
```

```
;
VTOP     equ      Ø                    VRAMの先頭アドレス
;
vstosw   macro    vramseg              VRAM STOSW マクロ定義
         MOV      CX,vramseg           CXにマクロパラメータでセグメント値を設定
         MOV      ES,CX                CXレジスタを介してESレジスタ値を設定
         MOV      CX,4Ø                ループ回数セット
         MOV      DI,BP                DIレジスタをBPで初期化
         REP      STOSW                ES:[DI]←AXをCX回繰り返す
         endm                          マクロ定義終了

SEEKLD:  ;SEEK Letter Data
         MOV      AH,Ø                 AH←0
         XCHG     AL,AH                AX←文字サーチ・コード番号×100H
         SHR      AX,1                 AX←AX÷2 …… コード番号×80Hに相当
         ADD      AX,2ØØØH             AX←AX+2000H
         RET                           リターン
;
MVPAIN:  ;MoVe and PAINt               ── 移動方向別消去および次座標計算
         PUSH     CX                   CXレジスタ値をスタックへ退避
         CMP      AL,UP                ALは上方向か?
         JE       UPAIN                上であればUPAINへ
         CMP      AL,DOWN              ALは下方向か?
         JE       DPAIN                下であればDPAINへ
         CMP      AL,LEFT              ALは左り方向か?
         JE       LPAIN                左り方向であればLPAINへ
;
RPAIN:   ;Right move PAINt
         MOV      BX,11ØH              BX←消去のサイズ
         CALL     CLPTXY               (CL, CH)からBXのサイズで消去する
         POP      CX                   スタックよりCXレジスタ値を復元
         INC      CL                   X座標を+方向へ更新:CL←CL+1
         RET                           リターン
;
UPAIN:   ;Up move PAINt
         ADD      CH,3                 Y方向を+3する:CH←CH+3
         MOV      BX,4Ø4H              BX←消去のサイズ
         CALL     CLPTXY               (CL, CH)からBXのサイズで消去する
         POP      CX                   スタックよりCXレジスタ値を復元
         DEC      CH                   Y座標を-方向へ更新:CH←CH-1
         RET                           リターン
LPAIN:   ;Left move PAINt
         ADD      CL,3                 X方向を+3する
         MOV      BX,11ØH              BX←消去のサイズ
         CALL     CLPTXY               (CL, CH)からBXのサイズで消去する
         POP      CX                   スタックよりCXレジスタ値を復元
         DEC      CL                   X座標を-方向へ更新:CL←CL-1
         RET                           リターン
```

```
;
DPAIN:  ;Down move PAINt
        MOV     BX,4Ø4H             BX←消去のサイズ
        CALL    CLPTXY              (CL,CH)からBXのサイズで消去する
        POP     CX                  スタックよりCXレジスタ値を復元
        INC     CH                  Y座標を＋方向へ更新：CH←CH+1
        RET                         リターン
;
SCLOC   equ     ØØ42H               SCore LOCation …… スコア表示座標
;
kasan   macro   op1,kasanchi        得点計算および表示マクロ定義開始
        MOV     AL,kasanchi         AL←加算値
        op1     AL,[BX]             AL←AL op1 DS：[BX] ： op1の演算を行う
        AAA                         ALの値をASCII加算補正
        PUSHF                       フラグをスタックへの退避
        MOV     [BX],AL             ALの値の保存
        PUSH    CX                  CXレジスタ値をスタックへ退避
        PUSH    BX                  BXレジスタ値をスタックへ退避
        CALL    DISPLE              ALで示される文字を(CL,CH)へ表示する
        POP     BX                  BXレジスタ値をスタックから復元
        POP     CX                  CXレジスタ値をスタックから復元
        SUB     CL,2                表示X座標を更新する(−2する)
        DEC     BX                  データ・アドレス更新(+1)する
        POPF                        フラグをスタックから復元
        endm                        マクロ定義の終了
;
TKETA   equ     6                   得点の桁数
;
DISPSC: ;DISPlay SCore              ── スコアの表示
        MOV     CX,offset SCLOC+2*TKETA   CX←1の位のスコア表示座標
        MOV     BX,offset SCOREL    BX←1,10の位の値が入っている
        kasan   ADD,DL              演算ADD,加算値DLでマクロ展開
        kasan   ADC,DH              演算ADC,加算値DLでマクロ展開
        XCHG    AX,CX               AX←→CX
        MOV     CX,TKETA-2          CX←点数の桁数分の繰り返し数−2
SCOREP: ;SCORE Print
        XCHG    AX,CX               AX←→CX
        PUSH    AX                  AXレジスタ値をスタックへ退避
        kasan   ADC,Ø               演算ADC,加算値DLでマクロ展開
        POP     AX                  AXレジスタ値をスタックから復元
        XCHG    AX,CX               AX←→CX
        LOOP    SCOREP              CX回ループ
        RET                         リターン
;
SCORE4  db      Ø                   得点表示ワークエリア4
SCORE3  db      Ø                   得点表示ワークエリア3
SCORE2  db      Ø                   得点表示ワークエリア2
SCORE1  db      Ø                   得点表示ワークエリア1
SCOREØ  db      Ø                   得点表示ワークエリア0
SCOREL  db      Ø                   得点表示ワークエリアLow
```

```
;
PAINWK  DB      7  DUP (?)                          PAINt Work area
;
MYWORK  DB      Ø  ;MY WORK area                    MY WORK area 移動方向
        DB      Ø                                   移動カウンタ
        DB      Ø,Ø                                 座標
        DB      Ø                                   残数
;
MYPT    equ     1  ;MY PaTtern number               MY PaTern number
;
MYMOVE: ;MY MOVE                                    —— 移動方向の決定 BX←MYWORK(移動方向)
        CALL    KEYCHK                              キーのチェック
        MOV     AL,CS:[BX]                          AL←移動方向
        OR      AL,AL                               ALは0か?
        JNE     NMYDIS                              0でなければNMYDISへ
        JMP     MYDISP                              0であればMYDISPへジャンプ
NMYDIS: INC     BX                                  BX更新
        MOV     DH,CS:[BX]                          DH←移動カウンタ値
        INC     BX                                  BX更新
        MOV     CL,CS:[BX]                          CL←X座標
        INC     BX                                  BX更新
        MOV     CH,CS:[BX]                          CH←Y座標
        PUSH    AX                                  AXレジスタ値をスタックへ保存
        CALL    GETCOL                              方向別の消去色を(COLOR)に入れる
        POP     AX                                  AXレジスタ値をスタックから復元
        PUSH    AX                                  AXレジスタ値をスタックへ保存
        CALL    CCHAN                               カウンタの増減値を方向別にALに求める
        MOV     BX,offset MYWORK                    BX←ワークエリア先頭アドレス
        INC     BX                                  BX←BX+1
        ADD     AL,CS:[BX]                          AL←AL+CS:[BX]
        AND     AL,3                                AL←0~3
        MOV     CS:[BX],AL                          CS:[BX]←AL
        POP     AX                                  AXレジスタ値をスタックから復元
        MOV     CX,word ptr [MYWORK+2]              CX←主人公の座標
        CALL    MVPAIN                              移動方向別の消去CXは次座標になる
        MOV     word ptr [MYWORK+2],CX              主人公の座標の更新
        MOV     AL,CS:[MYWORK+1]                    AL←移動カウンタ値
        OR      AL,AL                               ALは0か?
        JE      NMYDP2                              0であればNMYDP2へ
        JMP     MYDP2                               MYDP2へジャンプ
NMYDP2: ;Not MY DISPlay
        MOV     AL,CS:KEYDAT                        AL←キーデータ
        AND     AL,8ØH                              [SPACE]に対応するビットのチェック
        JNE     MYDISP                              [SPACE]が押されていればMYDISPへ
        CALL    BCTOIM                              (CL,CH)に対応する仮想迷路アドレスを求める
        MOV     AL,[BX]                             AL←[BX] …… 現在の道(ブロック)の色
        CMP     AL,7                                ALと7を比較する
        JE      MYDISP                              AL=7であればMYDISPへ
        INC     AL                                  パレットコードの更新
        MOV     [BX],AL
        MOV     DL,AL                               DL←AL
```

```
        MOV     DH,Ø                        DH←0
        PUSH    AX                          AXレジスタ値をスタックへ保存
        CALL    DISPSC                      スコアアップ
        POP     AX                          AXレジスタ値を復元
        MOV     CL,AL
        MOV     CH,Ø
        MOV     BX,offset PAINWK            BX←パレット・コード別のワークエリア
        DEC     BX                          (ペイントされていないマス数が入っている)
        ADD     BX,CX
        DEC     byte ptr [BX]
        JNE     MYDISP                      [BX]=0でなければMYDISPへ
        CMP     AL,7                        ALと7との比較
        JE      MYDISP                      AL=7であればMYDISPへ
        INC     BX                          BX←BX+1
        MOV     AL,[BX]                     AL←次のパレット番号の塗残し数
BONSCH: ;BONuS CHeck
        CMP     AL,Ø                        0はダミー,実際には道の総マス数がはいる
        JNE     MYDISP                      1マスでも塗られていればMYDISPへ
        ADD     SCORE2,CL                   ペイント完了した色番号×1000の
        XOR     DX,DX                       ボーナス得点を与える
        CALL    DISPSC
MYDISP: ;MY DISPlay
        MOV     AL,[MYWORK+1]               AL←移動カウンタ
MYDP2:  ;MY DisPlay2
        AND     AL,1                        AL←0,1となる
        ADD     AL,MYPT                     主人公のパターン1,2が交互に作られる
        MOV     CX,word ptr [MYWORK+2]      CX←座標
        CALL    DISP                        (CL,CH)にALを表示
        RET                                 リターン
;
CCHAN:  ;Counter CHANge                     ── 方向別にカウンタの増減値を求める
        CMP     AL,UP                       ALは上か?
        JE      SUBC                        上に等しければSUBCへ
        CMP     AL,LEFT                     ALは左か?
        JE      SUBC                        左に等しければSUBCへ
        MOV     AL,1                        下または右なら+1
        RET                                 リターン
SUBC:   ;SUBtract Counter
        MOV     AL,-1                       上または左なら-1
        RET                                 リターン
;
GETCOL: ;GET COLor number                   ── 消去に必要な色を(COLOR)に入れる
        MOV     DL,AL                       DL←AL
        CALL    BCTOIM                      BX←(CL,CH)に対する仮想迷路アドレス
        OR      DH,DH                       DHは0か?
        JE      GETCO2                      DH=0であればGETCO2へ
        MOV     AL,DL                       AL←DL:移動方向
        CMP     AL,DOWN                     移動方向は下か?
        JE      GETCO2                      下であればGETCO2へ
        CMP     AL,RIGHT                    移動方向は右か?
        JE      GETCO2                      右であればGETCO2へ
```

```
         MOV     CX,1            消去色を求める仮想迷路アドレスのオフセット値
         CMP     AL,LEFT         移動方向は左か？
         JE      GETCO1          左であれば GETCO1 へ
         MOV     CX,IMZX         消去色を求める仮想迷路アドレスのオフセット値
GETCO1:  ;GET COlor 1
         ADD     BX,CX           BX ← BX＋CX
GETCO2:  ;GET COlor 2
         MOV     AL,[BX]         AL ←[BX] …… 消去のパレット番号
         MOV     COLOR,AL        パレット番号の保存
         RET                     リターン
;
BCTOIM:  ;Cx TO Image Maze
         SHR     CL,1            CL ← CL÷2
         SHR     CL,1            CL ← CL÷2
         MOV     AL,CH         ┐
         ADD     AL,AL         │
         ADD     AL,AL         │
         AND     AL,ØFØH       │
         MOV     BL,AL         ├ CH が 4 の倍数でないときの余りをとる
         MOV     BH,Ø          │
         MOV     CH,BH         │
         ADD     BX,CX         ┘
         MOV     CX,offset IMAZE CX ←仮想迷路の先頭アドレス
         ADD     BX,CX           BX ← BX＋CX
         RET                     リターン
;
KEYCHK:  ;KEY CHeck
         MOV     AL,KEYDAT       AL ←キーデータ
         MOV     CH,AL           CH ← AL
         TEST    AL,UP           キーデータは UP である
         JE      KCK1            キーデータが UP でなければ KCK1 へ
         TEST    AL,DOWN         キーデータは DOWN である
         JE      KCK1            キーデータが DOWN でなければ KCK1 へ
         AND     AL,6            [8]，[2] キーがともに押されているのでキャンセルする
         MOV     CH,AL           CH ← AL
KCK1:    ;Key ChecK 1
         TEST    AL,LEFT         [6] が押されているか？
         JE      KCK2            押されていなければ KCK2 へ
         TEST    AL,RIGHT        [4] が押されているか？
         JE      KCK2            押されていなければ KCK2 へ
         AND     AL,9            [4]，[6] キーがともに押されているのでキャンセルする
         MOV     CH,AL           CH ← AL
KCK2:    ;Key CheCk 2
         MOV     BX,offset MYWORK ┐
         INC     BX               │
         MOV     AL,[BX]          ├ 移動カウンタ＝0 なら JUST へ
         OR      AL,AL            │
         JNE     NJUST            │
         JMP     JUST             ┘
```

```
NJUST:  ;Not JUST
        DEC     BX                      } AL←現在の移動方向
        MOV     AL,[BX]
        AND     AL,CH                   } 押されたと判定されたキーの値のなかに
        JE      $+3                       現在の移動方向が含まれていればリターン
        RET
        MOV     AL,[BX]                 AL←現在の移動方向
        CMP     AL,UP                   移動方向は上か？
        JE      DNCK                    上ならDNCKへ
        CMP     AL,DOWN                 移動方向は下か？
        JE      UPCK                    下ならUPCKへ
        CMP     AL,LEFT                 移動方向は左か？
        JE      RICK                    左ならRICKへ
LECK:   ;LEft CheckK
        TEST    CH,2                    [4]が押されたか？
        JNE     $+3                     押されていなければリターン
        RET                             リターン
        MOV     byte ptr [BX],LEFT      移動方向を左とする
        RET                             リターン
DNCK:   ;DowN CheckK
        TEST    CH,DOWN                 [2]が押されていなければリターン
        JNE     $+3                     押されていなければリターン
        RET                             リターン
        MOV     byte ptr [BX],DOWN      移動方向を下とする
        RET                             リターン
UPCK:   ;UP CheckK
        TEST    CH,UP                   [8]が押されたか？
        JNE     $+3                     押されていなければリターン
        RET                             リターン
        MOV     byte ptr [BX],UP        移動方向を上とする
        RET                             リターン
RICK:   ;RIght CheckK
        TEST    CH,RIGHT                [6]が押されたか？
        JNE     $+3                     押されていなければリターン
        RET                             リターン
        MOV     byte ptr [BX],RIGHT     移動方向を右とする
        RET                             リターン
;
JUST:   ;JUST counter=Ø
        PUSH    CX                      キーデータ(CH)をスタックへ退避
        INC     BX                      BX←BX+1
        MOV     CL,[BX]                 CL←X座標
        INC     BX                      BX←BX+1
        MOV     CH,[BX]                 CH←Y座標
        CALL    GETARR                  AL←そのブロックの座標データ：移動方向矢印
        SUB     BX,3                    BX←BX−3 …… MYWORK(移動方向)となる
        POP     CX                      スタックからキーデータ(CH)を復元
        MOV     CL,AL                   CL←AL …… 座標データ
        MOV     AL,CH                   AL←CH …… 押されたと判定されたキーデータ
        AND     AL,CL                   AL←AL & CL …… 移動可能なキーの値
        JNE     NNOMAT                  移動方向があればNNOMATへ
```

```
        JMP     NOMAT                       NOMATへジャンプ
NNOMAT: ;Not NOMAT
        XCHG    AL,AH                       AL ←→ AH
        MOV     AL,[BX]
        AND     AL,9                        現在方向＝左または右ならJUST1へ
        JE      JUST1
        XCHG    AL,AH                       AL ←→ AH
        MOV     byte ptr [BX],RIGHT         移動方向を右とする
        TEST    AL,RIGHT                    [6]のキーが押されているか？
        JE      $+3                         押されていなければ次をスキップ
        RET                                 リターン
        MOV     byte ptr [BX],LEFT          移動方向を左とする
        TEST    AL,LEFT                     [4]のキーが押されているか？
        JE      $+3                         押されていなければ次をスキップ
        RET                                 リターン
        MOV     byte ptr [BX],DOWN          移動方向を下とする
        TEST    AL,DOWN                     [2]のキーが押されているか？
        JE      $+3                         押されていなければ次をスキップ
        RET                                 リターン
        MOV     byte ptr [BX],UP            移動方向を上とする
        RET                                 リターン
JUST1:  ;JUST 1
        XCHG    AL,AH                       AL ←→ AH
        MOV     byte ptr [BX],UP            移動方向を上とする
        TEST    AL,UP                       [8]のキーが押されているか？
        JE      $+3                         押されていなければ次をスキップ
        RET                                 リターン
        MOV     byte ptr [BX],DOWN          移動方向を下とする
        TEST    AL,DOWN                     [2]のキーが押されているか？
        JE      $+3                         押されていなければ次をスキップ
        RET                                 リターン
        MOV     byte ptr [BX],LEFT          移動方向を左とする
        TEST    AL,LEFT                     [4]のキーが押されているか？
        JE      $+3                         押されていなければ次をスキップ
        RET                                 リターン
        MOV     byte ptr [BX],RIGHT         移動方向を右とする
        RET                                 リターン
NOMAT:  ;NO MATch
        MOV     [BX],AL                     (移動方向)← AL …… AL=0(停止)
        RET                                 リターン
;
GETARR: ;GET ARRow data                     ── ALに座標データを入れる
        PUSH    BX                          BXレジスタ値をスタックへ退避
        SHR     CL,1                        CL ← CL÷2
        SHR     CL,1                        CL ← CL÷2
        MOV     AL,CH                       AL ← CH
        ADD     AL,AL                       AL ← AL＋AL
        ADD     AL,AL                       AL ← AL＋AL
        MOV     BL,AL                       BL ← AL
        MOV     BH,Ø                        BH ← 0
        MOV     CH,BH                       CH ← 0
```

```
        ADD     BX,CX                   BX←AL×4+CL÷4
        MOV     CX,offset AMAZE         CX←座標データの先頭アドレス
        ADD     BX,CX                   BX←BX,CX
        MOV     AL,[BX]                 AL←[BX]
        POP     BX                      BXの値をスタックから取り出す
        RET                             リターン

        include LIST5-1.ASM             LIST5-1.ASMを取り込む
```

テスト5-2 テスト・プログラム(TEST5-2.ASM)

```
;
;****** TEST5-2 ********
;
CODE    segment                         命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,SS:STSEG

print   macro   string                  文字列出力マクロの定義
        LEA     DX,string               マクロパラメータのオフセットをDXへ
        MOV     AH,9                    ファンクションコール番号9 …… 文字列出力
        INT     21H                     ファンクションコール
        endm                            マクロ定義終了
;
PTEST:  ;Program TEST
        CLD                             ディレクション・フラグ・リセット
        MOV     AX,CS                   AX←CS
        MOV     DS,AX                   AXレジスタを介してDSにCSを格納
        print   CLEAR                   テキスト画面クリア
        CALL    DATLD                   パターン・データ・ロード
        CALL    GINIT                   グラフィック・システム初期化
        CALL    SETINT                  キーボード割り込みセット
        CALL    CLS                     グラフィック画面クリア
        CALL    MKIMZ                   仮想迷路を作る(周囲はFFH)
        CALL    MKAMZ                   矢印迷路を作る
        CALL    RMEDGE                  仮想迷路から周囲のFFHを取る
        CALL    DISPMZ                  迷路の表示
        XOR     AL,AL                   AL←0
        MOV     BX,offset MYWORK        BX←主人公のワークエリア
        MOV     CS:[BX],AL              移動方向←0
        INC     BX
        MOV     CS:[BX],AL              移動カウンタ←0
        INC     BX
        MOV     CS:[BX],AL              X座標←0
        INC     BX
        MOV     CS:[BX],AL              Y座標←0
```

```
TLOOP:  ;Test LOOP
        MOV     AX,ØCØØH        } キーボード・バッファ・クリア
        INT     21H             }
        MOV     AL,KPESC        STOP の情報を取り出す
        AND     AL,AL           AL は 0 か？ …… STOP が押されたか？
        JNE     TEND            STOP が押されたなら TEND へ
        CALL    MYMOVE          主人公の移動
        JMP     TLOOP           TLOOP へ
TEND:   ;Test END
        MOV     AH,41H          グラフィック停止コマンド
        INT     18H             ROM 内ルーチン・コール
        CALL    ORIINT          割り込み処理を初期状態へ戻す
        print   CSRON           カーソル・オン
        MOV     AX,ØCØØH        } キーボード・バッファ・クリア
        INT     21H             }
        MOV     AL,Ø            リターン・コード・ノーマル
        MOV     AH,Ø4CH         プロセスの終了
        INT     21H             ファンクションコール
;
lodinf  struc                   ロード情報構造体定義
CMD     db      Ø               ロード・コマンド
PASS    db      "           "   パス格納領域（11 文字分）
ENDSIN  db      Ø               パス・エンド・コード
LDADR   dw      Ø               ロード・アドレス
LDSEG   dw      Ø               ロード・セグメント
lodinf  ends                    構造体定義終了
;
LODTBL  lodinf <1,"MEIRO.DAT" ,Ø,Ø,PTNSEG>
        lodinf <1,"MOJI.DAT"  ,Ø,2ØØØH,PTNSEG >
        lodinf < >
;
        include LIST5-2.ASM     LIST5-2.ASM を取り込む
```

4. 追跡…サァ,追いかけよう!

テレビ・ドラマや映画を見ていると,かならず主人公を悩ますイジワルな人物が登場しますが,冷静に考えれば彼らの存在により物語が進行しているのです.さらに何も事件の起きない平和な場面ばかりでは,見ていても面白くも何ともないわけです.ウマイ役者であればあるほど,見るからに憎たらしい演技をし,まるでそれが本人の性格であるかのような錯覚を起こさせます.そのため,悪役というのはいつも悪役になるケースが多く,また見るほうもそのほうが安心して見られるということになります.これとは逆に,悪いことはできないというイメージが定着すると,どんなに演技がうまくても悪役は似合わなくなってしまうから不思議なものです.

ゲームの世界は,映画などに比べるとはるかに「小さな世界」ですから,このように見ただけでそのイメージが強烈に沸いてくる,ということはあまり考えられません.

そのため,1つのゲームのなかでプレイヤーに「憎たらしいヤツだ」とか,「バカなヤツだ」と思われるように,わかりやすい性格を付けてやる必要があります.主人公には,キー入力による制限で簡単に性格を付けてやれますが,悪役となる敵に対しては知恵を授けてやらなければなりません.

何だか難しいテーマのようになってきましたが迷路型ゲームにおいてどのようなときに敵が賢く見えるかを考えれば,答えは簡単明瞭です.それは,敵が主人公を追いかける,あるいは遠くから近づいて来る,ということができるかどうかです.これを知能と呼ぶには,あまりにもアツカマシイかもしれませんが,前章のゲームではデータによって移動方向を決めていただけですから,追跡をするということは,偉大なる知恵がついたといえるわけです.もし,これが主人公の次の動きを想定して動いてくれるというのであれば,本物の人工知能になれるのですが,実際には主人公のワークエリアにある座標を見て動くだけですから,悲しいかなイカサマの知能でしかありません.

イカサマの知能を,いかにして本物らしく見せるか,それがここでのテクニックということであり,また敵に与える性格なのです.そこで,3種類の敵に対して,次のような性格を付けて,プレイヤーをだましてみます.

敵のタイプ番号=0(パターン番号では3,4)　…… フラフラ
敵のタイプ番号=1(パターン番号では5,6)　…… 追いかけ
敵のタイプ番号=2(パターン番号では7,8)　…… 気まぐれ

フラフラは，乱数との組み合せでランダムに動くだけです．追いかけは，ここでの追跡ルーチンに従い主人公を追いかけます．気まぐれは，16ブロック移動するたびにフラフラと追いかけの動きを交互に繰り返していきます．ランダムに動くということは，座標データから1方向を選べばいいのですから，乱数を利用すれば簡単にできそうです．ということは，性格の違う3匹の敵がいるといっても，重要なのは追跡ルーチンを確立することだけになるわけです．

そこで，まずはどのように追いかけるのか，追跡の方針を決定しなければなりません．追跡とは，すなわち移動方向を一定の条件に基いて決めることですが，このゲームにおいては，その前提条件として「Uターンおよび停止はしない」ということにしてあります．ただし，迷路によっては袋小路があることも考えられるので，その際はUターンをすることにします．また，方向の変更があるのは，当然のことですが移動カウンター＝0の地点(座標データのある場所)だけになります．

この図5-7のなかで，優先方向の最後に「残り」というのがあります．これは基本的には行きたくない方向なので，たとえ2方向が残っていても優先権は付けず乱数に

1．座標データから，行ける方向数(矢印の数)を数える
2．数えた値が1の時は，座標データ通りの方向(袋小路)
3．数えた値が2の時は，反対方向でない方向(一本道)
　　―以上は，フラフラと共通―
4．座標データから，反対方向を除く(Uターンの禁止)
5．主人公の座標＝(BL,BH)，敵の座標＝(CL,CH)とする
6．方向の決定……反対方向を除いた座標データに，①②③の順に優先権をつけ，移動方向を決定する

主人公と敵との位置	CH≧BH,CL≧BL	CH≧BH,CL>BL	CH<BH,CL≧BL	CH<BH,CL<BL
Y X				
Y軸の差 > X軸の差 IB−HI　IC−LI	①　②　③ ↑　←　残り	①　②　③ ↑　→　残り	①　②　③ ↓　←　残り	①　②　③ ↓　→　残り
Y軸の差 ≦ X軸の差 IB−HI　IC−LI	①　②　③ ←　↑　残り	①　②　③ →　↑　残り	①　②　③ ←　↓　残り	①　②　③ →　↓　残り

注：残りの中での優先権は定めず，乱数との組み合わせで決定する

図5-7　追跡のルール

よってどちらかを選択するようにしているのです．ここでも，追跡にある程度の自由性を持たせているわけです．

この追跡のルールは，追跡としては最も基本的なパターンなのですが，迷路の形によっては図5-8のように同じ所をグルグル回ってしまうという欠点があります．

これは，仕方がないといってしまえばそれまでなのですが，知能的な動きからはあまりにもかけ離れています．いちばん簡単な解決方法は，このような動きが出ないような迷路にするということなのですが，最近は迷路にコンストラクション・セットを付ける場合も多くなっています．そのようなときに，知恵を与えた敵がこの程度の状態から抜け出せないのでは，すでにプレイヤーとの勝負に負けていることになり，作者として非常にクヤシイ思いがします．そこで，追跡ルーチンの最初に乱数を取り，少ない確率ではあるがランダムに動くルーチンにも分岐するようにするのです．こうすれば，何回かグルグル回れば，かならず抜け出るような動きをしてくれるので，いかにも敵が自分の頭で考えているように見えてきます．答えがわかると「ナーンだ!!」ということになってしまいますが，要はいかにして「……らしく見せるか」ですから，簡単なほどいいのです．ゲームとは，つきつめればキツネとタヌキの化かし合い，いやプレイヤーと作者との知恵比べみたいなものですからね．簡単なテクニックでプレイヤーをダマせれば，作る側の勝ちということです．

追跡の内容が理解できれば，このプログラムも意外と短く感じるかもしれません．

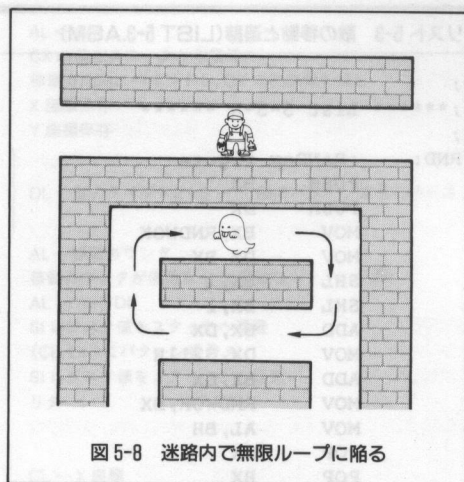



図5-8 迷路内で無限ループに陥る

結局，キー入力による方向決定にしても，この追跡ルーチンにしても，長く見えるのは方向あるいは位置別に，同じようなプログラムを作らなければならないためで，1つ1つはそれほどのことではないのです．案外，このあたりがマシン語を難しく思わせていた理由だったのかもしれません．

また，テスト5-3のプログラムでは，最初にMS-DOSのファンクションコールを利用して，時刻の秒の値を乱数の初期値としてワークエリアに取り入れ，動きに変化をつけています．

では，テスト・プログラムを実行してみましょう．どこに逃げても，アッというまに追いかけてくるはずです．このテストでは，敵の追いかけを1種類だけにしていますが，これを決めているのは敵のタイプ番号ですから，この値を0または2にすることにより，それぞれフラフラと気まぐれに変化します．性格によって動きがどのように違うか，試しに確認してみてください．

リスト 5-3 敵の移動と追跡(LIST 5-3.ASM)

```
;
;****** List 5-3-N ******
;
RND:     ;RaNDom figure
         PUSH     BX                  BXの値をスタックへ退避
         PUSH     DX                  DXの値をスタックへ退避
         MOV      BX,RNDWOK           BX←前回の乱数基数
         MOV      DX,BX               DX←BX
         SHL      BX,1                BX←BXの2倍
         SHL      BX,1                BX←さらに2倍… もとの数の4倍になる
         ADD      BX,DX               BX←BX+DX … もとの数の5倍になる
         MOV      DX,3211H            DX←3211H(適当な数値)
         ADD      BX,DX               BX←BX+DX
         MOV      RNDWOK,BX           RNDWOK←BX
         MOV      AL,BH               AL←BH
         POP      DX                  スタックからDXレジスタ値を復元
         POP      BX                  スタックからBXレジスタ値を復元
         RET                          リターン
;
RNDWOK   dw       1437                RaNDom figure WOrK area
;
tinfo    struc                        敵ワークエリア用構造体定義
         ETYPE     db 0                 敵のタイプ
         DIRECTION db 0                 移動方向
         COUNTER   db 0                 移動カウンタ
         XZAHYOU   db 0                 X座標
         YZAHYOU   db 0                 Y座標
tinfo    ends                         構造体定義の終わり

ENEMY    tinfo    <>                  敵ワークエリア3つ分確保
         tinfo    <>
         tinfo    <>
;
EMMOVE: ;EneMy MOVE
         MOV      AL,[SI].COUNTER     AL←移動カウンタ
         OR       AL,AL               AL=0か?
         JNE      $+5                 0でなければ次をスキップ
         CALL     EMJUST              新しい移動方向を決める
         MOV      CL,[SI].XZAHYOU     CL←X座標
         MOV      CH,[SI].YZAHYOU     CH←Y座標
         MOV      DH,[SI].COUNTER     DH←敵の移動カウンタ
         MOV      AL,[SI].DIRECTION   AL←敵の移動方向
         PUSH     CX                  CXの値をスタックへ退避
         CALL     GETCOL              移動方向別の消去(ペイント)色を(COLOR)へ
         MOV      AL,[SI].DIRECTION   AL←敵の移動方向
         MOV      CH,AL               CH←AL
         CALL     CCHAN               移動方向別に移動カウンタの増減値を求める
         ADD      AL,[SI].COUNTER   ┐
         AND      AL,3              ├ 移動後の移動カウンタ値
         MOV      [SI].COUNTER,AL   ┘
```

```
        MOV     AL,CH                          AL←CH
        POP     CX                             CXの値をスタックから復元
        CALL    MVPAIN                         移動方向別にペイントし,CXに次座標を求む
        MOV     [SI].XZAHYOU,CL                X座標保存
        MOV     [SI].YZAHYOU,CH                Y座標保存
        MOV     AL,[SI].ETYPE                ┐
        ADD     AL,AL                        │
        ADD     AL,3                         ├ DL←敵のタイプ×2+3…… 敵のパターン番号のベース
        MOV     DL,AL                        ┘
        MOV     AL,[SI].COUNTER                AL←移動カウンタ
        AND     AL,1                           移動カウンタが偶数なら0,奇数なら1
        ADD     AL,DL                          AL←AL+DL
        PUSH    SI                             SIレジスタ値をスタックへ退避
        CALL    DISP                           (CL,CH)にパターン番号AL
        POP     SI                             SIレジスタ値をスタックから復元
        RET                                    リターン
;
EMJUST: ;EneMy JUST counter=Ø
        MOV     CL,[SI].XZAHYOU                CL←X座標
        MOV     CH,[SI].YZAHYOU                CH←Y座標
        CALL    GETARR                         敵の現在地のデータをALに求める
        MOV     CH,7                           CH←7
        MOV     DL,AL                          DL←座標データ
        MOV     DH,AL                          DH←座標データ
        XOR     AL,AL                          AL←0
        MOV     CL,AL                          CL←0
EJLOOP: ;EmJust LOOP
        RCR     DH,1                         ┐
        ADC     AL,CL                        │
        DEC     CH                           ├ 矢印の総数
        JNE     EJLOOP                       ┘
        CMP     AL,3                           ALと3との比較
        JNB     NTAR12                         AL≧3ならNTAR12へ
        JMP     TAR12                          TAR12へジャンプ
NTAR12: ;Not TRA12
        MOV     AL,[SI].ETYPE                  AL←敵のタイプ
        OR      AL,AL                          AL=0か?
        JNE     NFREM1                         0でなければNFREM1へ
        JMP     FREEM                          FREEMへジャンプ
NFREM1: ;Not FREeM1
        DEC     AL                             AL=AL-1
        JNE     CANDF                          AL=0でなければCANDFへ
        JMP     BCHASE                         BCHASEへジャンプ
;
CANDF:  ;Chase AND Free
        MOV     BX,offset CFWORK               BX←追いかけとフラフラの回数カウンタ・アドレス
        DEC     byte ptr [BX]                  カウンタ値更新
CKCF:   ;Check AND Free
        JE      NOTJMP                         カウンタ値が0であればNOTJMPへ
        AND     byte ptr [CHJMP],1             CHJMP←CHJMP & 1
        JNE     NFREEM                         結果0でなければNFREEMへ
        JMP     FREEM                          FREEMへジャンプ
```

```
NFREEM: ;Not FREEM
        JMP     CHASE                     CHASE へジャンプ
NOTJMP: ;NOT JMP
        MOV     byte ptr [BX],16          新カウンタ値設定
        AND     byte ptr [CHJMP],1        CHJMP ← CHJMP & 1
        JE      CHANC                     結果 0 であれば CHANC へ
        MOV     byte ptr [CHJMP],Ø        CHJMP ← 0
        JMP     FREEM                     FREEM へジャンプ
CHANC:  ;CHANge to Chase
        MOV     byte ptr [CHJMP],1        CHJMP ← 1
        JMP     CHASE                     CHASE へジャンプ
;
CHJMP   db      Ø
;
CFWORK  db      Ø
;
TAR12:  ;Total Arrow = 1 or 2
        CMP     AL,1                      AL は 1 か？
        JNE     TAR2                      1 でなければ TRA2 へ
        MOV     [SI].DIRECTION,DL         座標データどうりの方向
        RET                               リターン
TAR2:   ;Total Arrow  = 2
        CALL    EROPP                     AL ← U ターンをしない移動可能方向
        MOV     [SI].DIRECTION,AL         移動方向保存
        RET                               リターン
;
EROPP:  ;ERase OPPosite direction
        MOV     AL,[SI].DIRECTION         AL ←現在の移動方向
        MOV     CH,AL                     CH ← AL
        AND     AL,9                      上下方向をマスク
        MOV     AL,6                      左右方向ビット=1
        JNE     EROPP1                    現在の移動方向が上下の場合は EROPP1 へ
        MOV     AL,9                      上下方向ビット=1
EROPP1: ;ERase OPPosite direction 1
        OR      AL,CH                     AL ← AL OR CH
        AND     AL,DL                     AL ← AL & DL
        RET                               リターン
;
FREEM:  ;FREE Move
        CALL    EROPP                     AL ← U ターンをしない移動可能方向
        MOV     CL,AL                     CL ← AL
SKFD:   ;SeeK Free Direction
        CALL    RND                       AL ←乱数
        AND     AL,CL                     乱数により移動可能方向を制限する
        JE      SKFD                      0 はカットする：0 であれば SKFD へジャンプ
        MOV     CL,AL                     CL ← AL
        AND     AL,CH                     AL ← AL & CH
        JE      $+3                       0 であれば次をステップ
        RET                               リターン
SKFD1:  ;SKFD 1
        CALL    RND                       AL ←乱数
        AND     AL,CL                     乱数により移動可能方向を制限する
```

```
        JP      SKFD1                       移動方向が1方向になるまでSKFD1へ
        MOV     [SI].DIRECTION,AL           移動方向保存
        RET                                 リターン
;
BCHASE: ;Before CHASE
        CALL    RND                         AL←乱数
        AND     AL,ØFH                      AL←0〜15
        JNE     CHASE                       0でなければCHASEへ
        JMP     FREEM                       1/16の確率でFREEMへ
CHASE:  ;CHASE
        CALL    EROPP                       AL←Uターンをしない移動可能方向
        MOV     BX,word ptr [MYWORK+2]      BX←座標
        MOV     CL,[SI].XZAHYOU             CL←X座標
        MOV     CH,[SI].YZAHYOU             CH←Y座標
        CMP     CH,BH                       CHとBHとの比較
        JNB     EPOSD                       CH>BHならEPOSDへ
        JMP     EPOSU                       EPOSUへジャンプ
;
EPOSD:  ;Enemy POSition Down
        CMP     CL,BL                       CLとBLとの比較
        JNB     EPOSDR                      CL>BLならEPOSDR
        JMP     EPOSDL                      EPOSDLへジャンプ
EPOSDR: ;Enemy POSition Down & Right
        MOV     DH,CH                       DH←CH
        SUB     DH,BH                       DH←CH−BH
        MOV     DL,CL                       DL←CL
        SUB     DL,BL                       DL←CL−BL
        MOV     BL,AL                       BL←AL
        CMP     DL,DH                       DLとDHとの比較
        JNB     EPDR1                       DL>DHならEPDR1へジャンプ
        MOV     AL,UP                       AL←UP
        AND     AL,BL                       上に移動可能か?
        JE      NOKDI1                      上に移動可能でなければ(=0)NOKDI1へ
        JMP     OKDIR                       OKDIRへジャンプ
NOKDI1: ;Not OKDIr 1
        MOV     AL,LEFT                     AL←LEFT
        AND     AL,BL                       左に移動可能か?
        JE      NOKDI2                      左に移動可能でなければ(=0)NOKDI2へ
        JMP     OKDIR                       OKDIRへジャンプ
NOKDI2: ;Not OKDIr 2
        JMP     TOSF1                       TOSF1へジャンプ
;
EPDR1:  ;EPosDR 1
        MOV     AL,LEFT                     AL←LEFT
        AND     AL,BL                       左に移動可能か?
        JE      NOKDI3                      左に移動可能でなければ(=0)NOKDI3へ
        JMP     OKDIR                       OKDIRへジャンプ
NOKDI3: ;Not OKDIr 3
        MOV     AL,UP                       AL←UP
        AND     AL,BL                       上に移動可能か?
        JE      NOKDI4                      左に移動可能でなければ(=0)NOKDI4へ
        JMP     OKDIR                       OKDIRへジャンプ
```

```
NOKDI4: ;Not OKDIr 4
        JMP     TOSF1                   TOSF1 へジャンプ
;
EPOSDL: ;Enemy POSition Down & Left
        MOV     DH,CH                   DH←CH
        SUB     DH,BH                   DH←CH−BH
        MOV     DL,BL                   DL←BL
        SUB     DL,CL                   DL←BL−CL
        MOV     BL,AL                   BL←AL
        CMP     DL,DH                   DL と DH との比較
        JNB     EPDL1                   DL>DH なら EPDL1 へ
        MOV     AL,UP                   AL←UP
        AND     AL,BL                   上に移動可能か？
        JE      NOKDI5                  上に移動可能でなければ(=0)NOKDI5 へ
        JMP     OKDIR                   OKDIR へジャンプ
NOKDI5: ;Not OKDIr 5
        MOV     AL,RIGHT                AL←RIGHT
        AND     AL,BL                   右に移動可能か？
        JE      NOKDI6                  右に移動可能でなければ(=0)NOKDI6 へ
        JMP     OKDIR                   OKDIR へジャンプ
NOKDI6: ;Not OKDIr 6
        JMP     TOSF1                   TOSF1 へジャンプ
EPDL1:  ;EPosDL 1
        MOV     AL,RIGHT                AL←RIGHT
        AND     AL,BL                   右に移動可能か？
        JE      NOKDI7                  右に移動可能でなければ(=0)NOKDI7 へ
        JMP     OKDIR                   OKDIR へジャンプ
NOKDI7: ;Not OKDIr 7
        MOV     AL,UP                   AL←UP
        AND     AL,BL                   上に移動可能か？
        JE      NOKDI8                  上に移動可能でなければ(=0)NOKDI8 へ
        JMP     OKDIR                   OKDIR へジャンプ
NOKDI8: ;Not OKDIr 8
        JMP     TOSF1                   TOSF1 へジャンプ
;
EPOSU:  ;Enemy POSition Up
        CMP     CL,BL                   CL と BL との比較
        JNB     EPOSUR                  CL>BL なら EPOSUR へ
        JMP     EPOSUL                  EPOSUL へジャンプ
EPOSUR: ;Enemy POSition Up & Right
        MOV     DH,BH                   DH←BH
        SUB     DH,CH                   DH←BH−CH
        MOV     DL,CL                   DL←CL
        SUB     DL,BL                   DL←CL−BL
        MOV     BL,AL                   BL←AL
        CMP     DL,DH                   DL と DH との比較
        JNB     EPUR1                   DL>DH であれば EPUR1
        MOV     AL,DOWN                 AL←DOWN
        AND     AL,BL                   下に移動可能か？
        JE      NOKDI9                  下に移動可能でなければ(=0)NOKDI9 へ
        JMP     OKDIR                   OKDIR へジャンプ
```

```
NOKDI9: ;Not OKDIr 9
        MOV     AL,LEFT                 AL ← LEFT
        AND     AL,BL                   左に移動可能か？
        JE      NOKDIA                  左に移動可能でなければ(=0)NOKDIA へ
        JMP     OKDIR                   OKDIR へジャンプ
NOKDIA: ;Not OKDIr A
        JMP     TOSF1                   TOSF1 へジャンプ
EPUR1:  ;EPosUR 1
        MOV     AL,LEFT                 AL ← LEFT
        AND     AL,BL                   左に移動可能か？
        JE      NOKDIB                  左に移動可能でなければ(=0)NOKDIB へ
        JMP     OKDIR                   OKDIR へジャンプ
NOKDIB: ;Not OKDIr B
        MOV     AL,DOWN                 AL ← DOWN
        AND     AL,BL                   下に移動可能か？
        JE      NOKDIC                  下に移動可能でなければ(=0)NOKDIC へ
        JMP     OKDIR                   OKDIR へジャンプ
NOKDIC: ;Not OKDIr C
        JMP     TOSF1                   TOSF1 へジャンプ
;
EPOSUL: ;Enemy POSition Up & Left
        MOV     DH,BH                   DH ← BH
        SUB     DH,CH                   DH ← BH−CH
        MOV     DL,BL                   DL ← BL
        SUB     DL,CL                   DL ← DL−CL
        MOV     BL,AL                   BL ← AL
        CMP     DL,DH                   DL と DH との比較
        JNB     EPUL1                   DL>DH なら EPUL1 へ
        MOV     AL,DOWN                 AL ← DOWN
        AND     AL,BL                   下に移動可能か？
        JNE     OKDIR                   下に移動可能ならば(≠0)OKDIR へ
        MOV     AL,RIGHT                AL ← RIGHT
        AND     AL,BL                   右に移動可能か?
        JNE     OKDIR                   右に移動可能ならば(≠0)OKDIR へ
        JMP     TOSF1                   TOSF1 へジャンプ
EPUL1:
        MOV     AL,RIGHT                AL ← RIGHT
        AND     AL,BL                   右に移動可能か？
        JNE     OKDIR                   右に移動可能ならば(≠0)OKDIR へ
        MOV     AL,DOWN                 AL ← DOWN
        AND     AL,BL                   下に移動可能か？
        JNE     OKDIR                   下に移動可能ならば(≠0)OKDIR へ
TOSF1:  ;TO SkFd 1
        MOV     CL,BL                   CL ← BL
        JMP     SKFD1                   SKFD1 へジャンプ
OKDIR:  ;OK DIRection
        MOV     [SI].DIRECTION,AL       移動方向保存
        RET                             リターン

        include LIST5-2.ASM             LIST5-2.ASM を取り込む
```

テスト5-3 テスト・プログラム(TEST 5-3.ASM)

```
;
;****** TEST5-3 ********
;
CODE    segment                             命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,SS:STSEG

print   macro   string                      文字列出力マクロの定義
        LEA     DX,string                   マクロパラメータのオフセットをDXへ
        MOV     AH,9                        ファンクションコール番号9 …… 文字列出力
        INT     21H                         ファンクションコール
        endm                                マクロ定義終了
;
PTEST:  ;Program TEST
        CLD                                 ディレクション・フラグ・リセット
        MOV     AX,CS                       AX←CS
        MOV     DS,AX                       AXレジスタを介してDSにCSを格納
        print   CLEAR                       テキスト画面クリア
        CALL    DATLD                       パターン・データ・ロード
        CALL    GINIT                       グラフィック・システム初期化
        CALL    SETINT                      キーボード割り込みセット
        MOV     byte ptr RNDWOK,5           乱数の初期値セット
        CALL    CLS                         グラフィック画面クリア
        CALL    MKIMZ                       仮想迷路を作る(周囲はFFH)
        CALL    MKAMZ                       矢印迷路を作る
        CALL    RMEDGE                      仮想迷路から周囲のFFHを取る
        CALL    DISPMZ                      迷路の表示
        XOR     AL,AL                       AL←0
        MOV     BX,offset MYWORK            BX←主人公のワークエリア
        MOV     CS:[BX],AL                  移動方向←0
        INC     BX                          BX←BX+1
        MOV     CS:[BX],AL                  移動カウンタ←0
        INC     BX                          BX←BX+1
        MOV     CS:[BX],AL                  X座標←0
        INC     BX                          BX←BX+1
        MOV     CS:[BX],AL                  Y座標←0
        MOV     SI,offset ENEMY             敵ワークエリアの先頭アドレス
        MOV     [SI].ETYPE,1                敵のタイプNO.1
        MOV     [SI].DIRECTION,LEFT         方向左
        MOV     [SI].COUNTER,Ø              カウンタ0
        MOV     [SI].XZAHYOU,Ø              X座標0
        MOV     [SI].YZAHYOU,44             Y座標44
TLOOP:  ;Test LOOP
        MOV     AX,ØCØØH                  } キーボード・バッファ・クリア
        INT     21H                       }
        MOV     AL,KPESC                    AL←STOPの情報
        AND     AL,AL                       [STOP]が押されたか?
        JNE     TEND                        押されていればTENDへ
        CALL    EMMOVE                      敵を移動
```

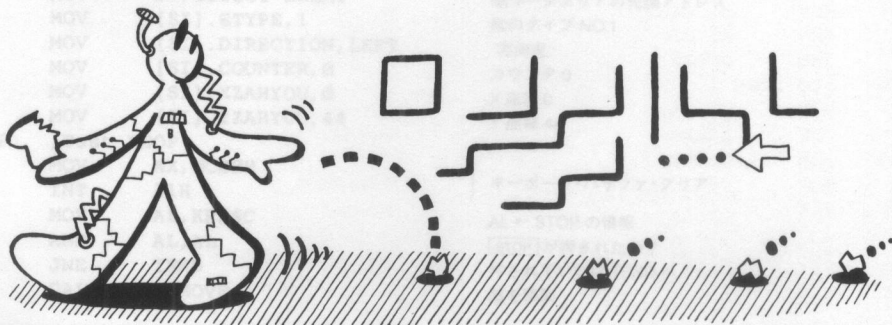
```
        PUSH    SI                      SI値をスタックへ保存
        CALL    MYMOVE                  主人公を移動
        POP     SI                      SI値をスタックから復元
        JMP     TLOOP                   TLOOPへ
TEND:   ;Test END
        MOV     AH,41H                  グラフィック停止コマンド
        INT     18H                     ROM内ルーチン・コール
        CALL    ORIINT                  割り込み処理を初期状態へ戻す
        print   CSRON                   カーソル・オン
        MOV     AX,ØCØØH               }
        INT     21H                    } キーボード・バッファ・クリア
        MOV     AL,Ø                    リターン・コード・ノーマル
        MOV     AH,Ø4CH                 プロセスの終了コマンド・セット
        INT     21H                     ファンクション・コール
;
lodinf  struc                           ロード情報構造体定義
CMD     db      Ø                       ロード・コマンド
PASS    db      "           "           パス格納領域（11文字分）
ENDSIN  db      Ø                       パス・エンド・コード
LDADR   dw      Ø                       ロード・アドレス
LDSEG   dw      Ø                       ロード・セグメント
lodinf  ends                            構造体定義終了
;
LODTBL  lodinf <1,"MEIRO.DAT",Ø,Ø,PTNSEG>
        lodinf <1,"MOJI.DAT" ,Ø,2ØØØH,PTNSEG >
        lodinf < >
;
        include LIST5-3.ASM             LIST5-3.ASMを取り込む
```

5. 完成…メッセージや音を付ける

プログラムに限らず，完成直前の状態というのは，だれでも一番胸がワクワクするものです．これは苦労が多いものほどうれしさも多く，できることならいつまでも完成直前のままにしておきたい，などという人もいるくらいです．しかし，ゲーム・プログラムのように完成後に楽しみがある場合は，そんな悠長なことは言ってられませんね．一刻も早く仕上げて，遊んでみたいと思うのが人情というものです．

しかし，長いプログラムを自分で作ると，この完成直前の期間が一番長く感じられ，イヤなものになってしまいます．その原因は，言わずと知れたバグという，コンピュータにとって生まれたときから永遠に続く，運命共同体とも言えるものがあるからです．とくに大きな作品になればなるほど，この時期になって出てくるオカシナ現象に悩まされるようになり，最後には折角自分で遊ぼうと思って作ったゲームなのに，見るのもイヤということになってしまうのです．そういうことが，わかっていながらまた次の作品を作りたくなってくるのは，やはり一種のコンピュータ病なのかもしれません．

本書のプログラムは，そういうことが起きないように，すでに十分に試験走行をしてありますので，安心してください．では最後に次の4つの内容を付加して本章のプログラムを完成させましょう．

1. 文字連続表示ルーチン …… 前回と同じ
2. 道の数を数えるルーチン …… ペイントされる道(ブロック)の総数
3. 全敵移動ルーチン …… 3種類の敵を動かす
4. 衝突判定ルーチン …… 前回と同じ

たったこれだけですから，説明を必要とするほど複雑なものはなく，プログラムを見れば一目瞭然だと思います．どちらかというと，ここではテストの部分の質を高めてあります．

まずノン・グラフィック・ルーチンのウェイトとして無駄命令ではなく音楽ルーチン(MUSIC)を用いてみました．本当は，完全な音楽を移動用として付けたかったのですが，プログラムの長さと音楽的な能力の問題から，ここでは主人公の移動カウンターを利用し，カウンター番号に合わせてソ・ラ・シ・ドの音を出しているだけです．気になる方は，音楽専用のカウンターとデータさえ用意すれば，まったく同じ方法でBGMが出せるようになりますから，ぜひチャレンジしてみてください．また，移動がないときには音が出ないようにしていますが，全体の速度が変化しないようにするには，その分ウェイトを入れてやらなければなりません．そこで，同じ音楽ルーチンから休符を利用して，これをウェイトとして使っています．このように，休符は単なるウェイトとしての役目も果たせますから，無駄命令の代わりに利用するとキメの細かなウェイトができます．

さらに，ゲーム完成後にメッセージを入れたり，ゲームが終わってもMS-DOSへ無造作に戻さず，SPACEで再ゲームができるようにした点などが，前回のものに比べると進歩していると言えます．しかし，いくら進歩したといっても，これだけでは本物のゲームと言えないのは，だれが見ても明らかです．次の6章では，もう少し工夫とアイデアを凝らし，本書最後のゲームとして市販の商品に負けないような大作を作っていきます．

リスト5-4　ペンキ・ボーイの仕上げ(LIST5-4.ASM)

```
;
;****** List 5-4-N ******
;
MSGPRN: ;MeSsaGe PRiNt
        MOV     AL,[BX]          AL←DS:[BX]
        OR      AL,AL            AL=0か？
        JNE     $+3              0に等しければリターン
        RET                      リターン
        CMP     AL,' '           AL=' 'か？
        JNE     MSG2             等しくなければMSG2へ
        MOV     AL,'Ø'+1Ø        AL←30H+10…空白を表す
MSG2:   ;MeSsaGe print-2
        SUB     AL,'Ø'           AL←AL−30H
        CMP     AL,11            AL<11か？
        JB      MSG1             であればMSG1へ
        SUB     AL,6             AL>11なら−6の補正
```

```
MSG1:   ;MeSsaGe print-1
        PUSH    CX                      スタックへCXレジスタ値を退避
        PUSH    BX                      スタックへBXレジスタ値を退避
        CALL    DISPLE                  (CL, CH)よりALを表示 …… 文字・数字・空白
        POP     BX                      スタックからBXレジスタ値を復元
        POP     CX                      スタックからCXレジスタ値を復元
        ADD     CL,2                    CL←CL+2
        INC     BX                      BX←BX+1
        JMP     MSGPRN                  MSGPRNへジャンプ
;
CPROAD: ;Count Paintable ROAD
        PUSH    SI                      SIをスタックへ退避
        PUSH    DI                      DIをスタックへ退避
        MOV     SI,offset PAINWK        道のブロック数がはいるワークエリア
        MOV     byte ptr [SI],Ø         DS:[SI]←0
        MOV     DI,offset IMAZE         DI←仮想迷路の先頭アドレス
        MOV     DX,CS                   DX←CS
        MOV     ES,DX                   ES←DX
        MOV     AL,Ø                    AL←0
        MOV     CX,192                  CX←192 …… 迷路の総ブロック数
CPRLP:  ;CPRoadLooP
        SCASB                           ┐
        JNE     SKIPC                   │
        INC     byte ptr [SI]           │ [SI]←道の部分の総ブロック数
SKIPC:  ;SKIP Count                     │
        LOOP    CPRLP                   ┘
        MOV     DI,SI                   ┐
        INC     DI                      │
        MOV     CX,6                    │ 7色分のPAINWKすべてに,[SI]の値を入れる
        REP     MOVSB                   ┘
        POP     DI                      スタックからDI値を復元
        POP     SI                      スタックからSI値を復元
        RET                             リターン
;
EMMVAL: ;EneMy MoVe ALl                 ┐
        MOV     SI,offset ENEMY         │
        MOV     CX,3                    │
EMALP:  ;EmMvAl LooP                    │
        PUSH    SI                      │
        PUSH    CX                      │
        CALL    EMMOVE                  │ 3タイプの敵をそれぞれ1コマ移動する
        POP     CX                      │
        POP     SI                      │
        ADD     SI,type ENEMY           │
        LOOP    EMALP                   │
        RET                             ┘
;
MYCHK:  ;MY CHeck
        MOV     BX,offset ENEMY         BX←敵1のX座標を示すワークエリア
        ADD     BX,3                    BX←BX+3
        MOV     CX,3                    CX←敵の総数
```

```
MCLOOP: ;MyChk LOOP
        MOV     AL,[MYWORK+2]
        SUB     AL,[BX]
        ADD     AL,2          主人公のＸ座標－敵のＸ座標＋2≧5 なら NCRASH へ
        CMP     AL,5
        JNB     NCRASH
        MOV     AL,[MYWORK+3]
        INC     BX
        SUB     AL,[BX]       主人公のＹ座標－敵のＹ座標＋2≧5 ならリターン
        DEC     BX            (キャリー・フラグが衝突のサイン)
        ADD     AL,2
        CMP     AL,5
        JNB     NCRASH
        RET                   リターン
NCRASH: ;No CRASH
        ADD     BX,5          BX ← BX＋5
        LOOP    MCLOOP        CX 回ループ
        RET                   リターン

        include LIST5-3.ASM   LIST5-3.ASM を取り込む
```


テスト 5-4　テスト・プログラム(TEST 5-4.ASM)

```
;
;******  TEST5-4 ********
;
CODE    segment               命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,SS:STSEG

print   macro   string        文字列出力マクロの定義
        LEA     DX,string     マクロパラメータのオフセットを DX へ
        MOV     AH,9          ファンクションコール番号 9 …… 文字列出力
        INT     21H           ファンクションコール
        endm                  マクロ定義終了
;
PTEST:  ;Program TEST
        CLD                   ディレクション・フラグ・リセット
        MOV     AX,CS         AX ← CS
        MOV     DS,AX         AX レジスタを介して DS に CS を格納
        print   CLEAR         テキスト画面クリア
        CALL    DATLD         パターン・データ・ロード
        CALL    GINIT         グラフィック・システム初期化
        CALL    SETINT        キーボード割り込みセット
        MOV     byte ptr RNDWOK,5   乱数初期値セット
        CALL    CLS           グラフィック画面クリア
        CALL    MKIMZ         仮想迷路を作る(周囲は FFH)
```

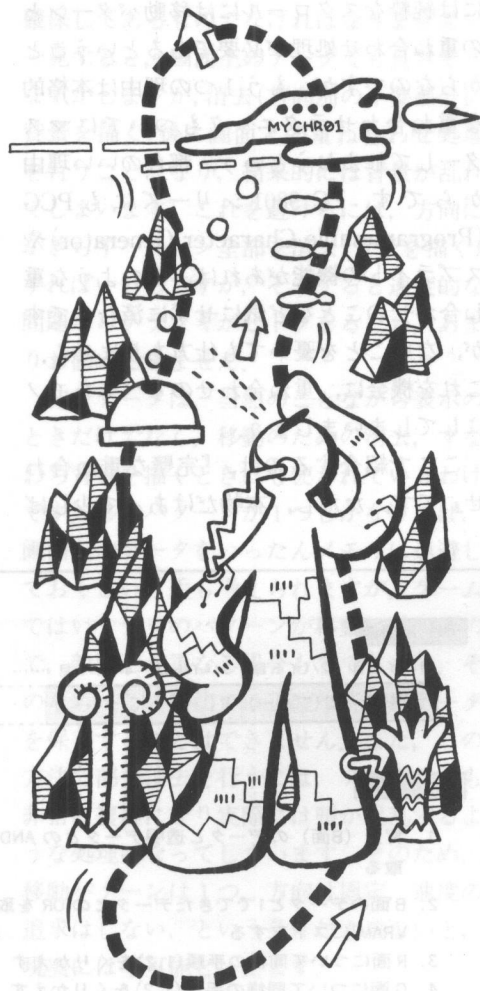
```
        CALL    MKAMZ                       矢印迷路を作る
        CALL    RMEDGE                      仮想迷路から周囲の FFH を取る
        CALL    DISPMZ                      迷路の表示
        CALL    CPROAD                      道の総ブロック数を数える
        MOV     AL,PAINWK                   AL ← 道の総ブロック数
        MOV     byte ptr BONSCH+1,AL        BONSCH＋1 ← AL：総ブロック数
        MOV     AL,16                       AL ← 16：追いかけとフラフラのカウンタ
        MOV     CFWORK,AL                   カウンタ値を初期化
        XOR     AX,AX                       AX ← 0
        MOV     BX,offset SCORE4          ┐
        MOV     [BX],AX                   │
        MOV     [BX+2],AX                 │ スコアの初期化と表示
        MOV     [BX+4],AX                 │
        MOV     DX,AX                     │
        CALL    DISPSC                    ┘
        MOV     CX,MANPOS                 ┐
        MOV     AL,1                      │ 右上部に主人公表示
        CALL    DISP                      ┘
        MOV     AL,3                      ┐ 主人公の初期数設定
        MOV     [MYWORK+4],AL             ┘
;
TEST1:  ;TEST 1
        XOR     AX,AX                     ┐
        MOV     BX,offset MYWORK          │ 主人公のワークエリア初期化
        MOV     [BX],AX                   │
        MOV     [BX+2],AX                 ┘
        MOV     SI,offset EMINIT          ┐
        MOV     DI,offset ENEMY           │
        MOV     CX,CS                     │ 敵のワークエリア初期化
        MOV     ES,CX                     │ EMINT を EMWORK へ転送する
        MOV     CX,3 * type ENEMY         │
        REP     MOVSB                     ┘
        MOV     AL,[MYWORK+4]             ┐
        MOV     CX,RESLOC                 │ 主人公の残数表示
        CALL    DISPLE                    ┘
;
TEST2:  ;TEST 2
        MOV     AX,ØCØØH                  ┐ キーボード・バッファ・クリア
        INT     21H                       ┘
        CALL    EMMVAL                      敵の移動
        CALL    MYMOVE                      主人公の移動
        MOV     BX,offset MYWORK          ┐
        XOR     AL,AL                     │ 主人公の移動方向＝0 なら NOMSC へ
        OR      AL,[BX]                   │
        JE      NOMSC                     ┘
        INC     BX                        ┐
        MOV     AL,[BX]                   │
        ADD     AL,AL                     │ DX ← 主人公の移動カウンタ値×3
        ADD     AL,[BX]                   │
        MOV     DL,AL                     │
        MOV     DH,Ø                      ┘
```

```
        MOV     BX,offset MMDATA        } BX ← DX+MMDATA …… 移動音データ・アドレス
        ADD     BX,DX                   }
        JMP     CMUSIC                  CMUSIC へジャンプ
NOMSC:  ;NO MuSiC
        MOV     BX,offset MMDATA        BX ← MMDATA 先頭アドレス
        ADD     BX,12                   BX ← BX+12
CMUSIC:
        CALL    MUSIC                   音楽演奏をする
        MOV     AL,KEYDAT               AL ← キー情報
        TEST    AL,80H                  [SPACE] のチェック
        JE      TMYCHK                  [SPACE] が押されていれば TEST2 へ
        JMP     TEST2                   TEST2 へジャンプ
TMYCHK: CALL    MYCHK                   } 主人公と敵が衝突していれば MYDEAD へ
        JB      MYDEAD                  }
        MOV     AL,[PAINWK+6]           }
        OR      AL,AL                   } カラー番号 7 のペイント残数≠0 なら TEST2 へ
        JE      NTEST2                  }
        JMP     TEST2                   }
NTEST2: JMP     WINNER                  WINNER へジャンプ
MYDEAD: ;MY DEAD
        MOV     BX,offset DMDATA        } 衝突音
        CALL    SND1                    }
        MOV     BX,offset MYWORK        }
        ADD     BX,4                    } 主人公の残数を－1 し 0 なら GOVER へ
        DEC     byte ptr [BX]           }
        JE      GOVER                   }
        CALL    DISPMZ                  迷路を表示
        JMP     TEST1                   TEST1 へ
GOVER:  ;Game OVER
        XOR     AL,AL                   }
        MOV     CX,RESLOC               } 残数表示位置に 0 を表示
        CALL    DISPLE                  }
        MOV     BX,offset GOMSG         BX ←「GAME OVER」の文字列アドレス
        JMP     MSG                     MSG へジャンプ
WINNER: ;WINNER
        MOV     BX,offset WMSG          BX ←「CONGRATULATIONS」の文字列アドレス
MSG:    ;MeSsaGe
        MOV     CX,100EH                } (0EH,10H) より文字列を表示
        CALL    MSGPRN                  }
        MOV     BX,offset PSMSG         }
        MOV     CX,2009H                } 「PRESS SPACE」の表示
        CALL    MSGPRN                  }
WLOOP:  ;Waiting LOOP
        MOV     AL,KEYDAT               AL ← キー情報
        TEST    AL,80H                  [SPACE] のチェック
        JE      WLOOP                   [SPACE] が押されていなければ WLOOP へ
        JMP     PTEST                   PTEST へジャンプ
;
GOMSG:  ;Game Over MeSsage
        db      'G A M E '
        db      'O V E R ',0
```

```
WMSG:   ;Winner's MeSsaGe
        db      '  CONGRAT'
        db      'ULATIONS ',Ø
PSMSG:  ;Press Space MeSsaGe
        db      ' PRESS SPACE '
        db      'TO REPLAY ',Ø
DMDATA: ;Dead Music DATA
        db      2Ø,4Ø,1Ø,Ø
        db      2Ø,8Ø,1Ø,Ø
        db      2Ø,12Ø,1Ø,Ø
        db      4Ø,16Ø,255,Ø,Ø
MMDATA: ;Move Music DATA
        db      22H,87H,Ø
        db      24H,78H,Ø
        db      28H,6BH,Ø
        db      2AH,65H,Ø
        db      14H,Ø,Ø
EMINIT: ;EneMy INITial data
        db      Ø,UP,Ø,6Ø,Ø
        db      1,LEFT,Ø,Ø,44
        db      2,DOWN,Ø,6Ø,44
;
MANPOS  equ     ØF44H                       MAN's POSition    残数左の主人公表示座標
RESLOC  equ     1Ø4AH                       REStnumberLOCation 残数表示座標
;
lodinf  struc                               ロード情報構造体定義
CMD     db      Ø                           ロード・コマンド
PASS    db      "           "               パス格納領域（11 文字分）
ENDSIN  db      Ø                           パス・エンド・コード
LDADR   dw      Ø                           ロード・アドレス
LDSEG   dw      Ø                           ロード・セグメント
lodinf  ends                                構造体定義終了
;
LODTBL  lodinf <1,"MEIRO.DAT",Ø,Ø,PTNSEG>
        lodinf <1,"MOJI.DAT" ,Ø,2ØØØH,PTNSEG >
        lodinf < >
;
        include LIST4-2.ASM                 LIST4-2.ASM の取り込み
        include LIST5-4.ASM                 LIST5-4.ASM の取り込み
```

章 ●スクロール・ゲーム

●リアルタイムゲームについて，これまでにインベーダー型，そしてパックマン型と，そのテクニックを次々と撃破してきたわけですが，最後にもう１つのタイプであるゼビウス型ゲームがあなたを待っています．何といっても，スクロールゲーム大流行の本家であり，神話までできたと言われるほどのゲームですから，本書でもその存在を無視するわけにはいきません．そのわりに，PC-9801シリーズ用のスクロールゲームは数多く出ているとは言えません．本来ならば，「猫も杓子もスクロール……」とばかりに，アチコチのソフトメーカーから商品化されてもよさそうなのですが，いったいどうしたのでしょうか……？

●このスクロールゲームは，マップやキャラクタ，敵の動きなど，あなた自身が作れるように，コンストラクション・セットとして構成してあります．

1. 重ね合わせ…もはや一般教養です

これから，画面スクロールというテーマに進むハズなのに，ここに出てきたタイトルは何と『重ね合わせ』です．これは，1つには純粋なスクロールには移動パターンとの重ね合わせ処理が必要であるということからなのですが，もう1つの理由は本格的な重ね合わせテクニックもついでにマスターしてしまおうという，都合のいい理由からです．PC-9801シリーズにもPCG (Programmable Character Generator)やスプライトの機能があれば，このような重ね合わせのことなど気にせずに済むのですが，ないことを憂いても仕方ありません．これを機会に，重ね合わせのすべてをモノにしてしまいましょう．

ここで紹介するのは，『完璧な重ね合わせ』です．ただし，本物だけあって少しばかりめんどうですし，パターン・データもこれまでとは違い『完璧な重ね合わせ』専用のものが必要になります．

その上，パターンを移動させるには，背景となっているB, R, Gの各面のデータを画面とは別に持つ必要があります．工夫すれば，これはデータそのものではなく，背景のパターン番号を示す画面データでもできないことはありませんが，プログラム的にはより複雑になります．

これらのことを頭に入れた上で，『完璧な重ね合わせ』のデータ構成，およびその手順を見てみることにいたしましょう(図6-1参照)．

まず，このような特殊なデータを作る方法ですが，これは，Appendixのパターン・エディタを使えば簡単に作成することがで

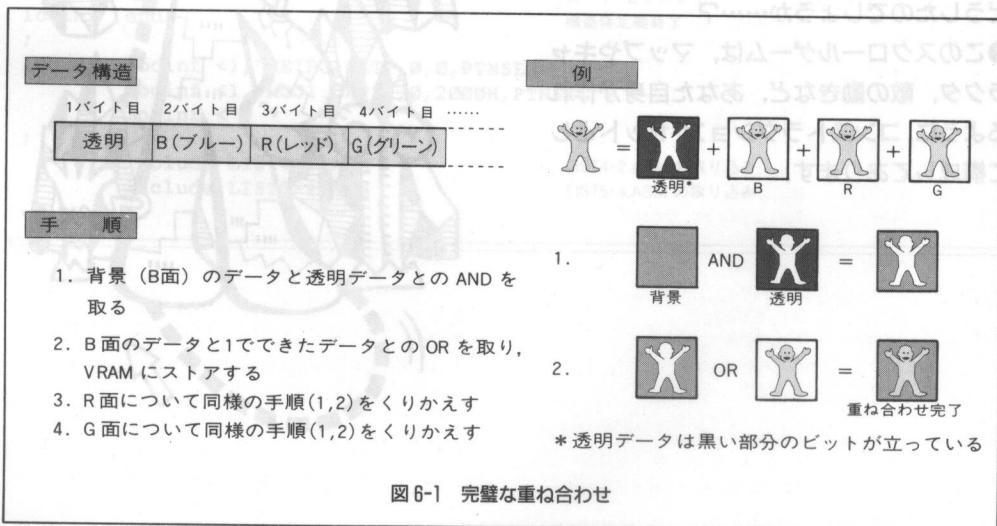



図6-1　完璧な重ね合わせ

きます．このときのデータはコマンドＤでT1BRGを選択してください．

本書では，PC-9801シリーズに共通して使えるB,R,Gの画面のみを使っているため，パレットとしては0~7番まであるわけですが，これにもう1つ，透明(背景)色という仮定の色を設定しています．つまり，透明(背景)を出したい部分をパレット番号の8(パターン・エディタのパレット番号)で描いておき，ソフト上で透明の処理，すなわち背景を表示するようにすればいいのです．

これで，希望どおりのデータができることになります．次は重ね合わせの手順です．図6-1の例で確認すると，最初に背景とのANDを取り，できたデータとその面のパターン・データとのORを取るという作業を，B·R·G各面について行えば，背景とパターンが完全に重なるというわけです．

さて，ここで示した方法は，PC-9801シリーズ共通に通用する方法ですが，最近では，EGCなどの新しい機能が加わりましたから，それらが使える局面では考え方が異なってきます．このEGCについては，いろいろおもしろいテクニックなどがあり，ゲーム制作にはうってつけなのですが，使用できる機種も限られるため，本書では全シリーズに通用する方法を取ることにしました．なお，EGCに関してはいずれ何らかの機会に詳述したいと思っています．もちろん，本書のパターン·エディタにはEGC用のデータを作るための機能も用意してありますが……．

さて，背景とパターンをただ重ね合わせるだけであれば，プログラムにするのもそれほど難しいことではないし，ANDを取るのも画面(VRAM)上のデータでいいのですが，たいへんなのは重ね合わせたパターンを移動させるときです．この場合，透明データとANDを取るのは，実際に画面上にあるデータではなく，別のメモリに確保してあるものでなければなりません．一見すると，画面上のデータでも良さそうな気がしますが，消去(方向別の不要部分に背景を描く)後に画面との重ね合わせ処理を行うことになり，結果的には背景が乱れてしまいます．これを避けるには，方向にかぎらずパターン全部を消去(背景を描く)すればいいのですが，そうすると速度的な問題からチラツキがヒドクなるため，あまりお勧めできません．

背景データは，当然のことながら表示のときだけでなく，移動のための消去，すなわち背景を描くときにも使われているわけです．移動パターンが1つしかなければ，画面上のデータをいったんメモリに退避しておくという手も考えられますが，ゲームではいくつものパターンが移動しているので，敵同士が重なる場合もでてきます．そのため，この方法では正確な背景のデータを保存することはできません．また，この方法で部分消去を行うには，プログラムも非常に複雑になり実際には頭が混乱するような処理になってしまいます．そのため，移動パターンは1つ，方向は固定，速度の追求はしない，という条件付きでないと，現実には不可能といえます．

以上のような理由から，どうしても画面上の背景データをどこかに確保しておかければならないということになるのです．

ところで，PC-9801 シリーズのVRAMは，都合のいいことに，640×400で表と裏の2画面分あります．ですから，この両方の画面にまったく同じ背景を描くようにするのです．

パターンを移動するときには，この裏画面との演算をほどこした結果を表画面に書き込めば，さきほどの問題は解決というわけです．

また，PC-9801 シリーズには 640×400で1画面しかVRAMを持っていない機種もありますが，そのときには640×200モードを選択してください．

このときのプログラムは，VRAMをアクセスしている部分を200ライン用に改造することになります．パターン表示プログラムでは，データを1ラインおきに表示するようにするとパターンエディタのデータがそのまま使えます．

リスト 6-1では，プログラムの動きが，わかりやすいように，裏画面に青色の背景を，表画面に緑色の背景を，あらかじめ描くようにプログラムしてありますが，実際には表と裏がまったく同じ絵となるのは言うまでもありません．

プログラムのテストに際しては，重ね合わせ専用のパターン・データ(コマンドDでT1BRGを選択)を用いないと，その意味がありませんから，巻頭口絵4のパターンを参考にし専用のデータを作成してから，実行してください．なお，パターンのパス名は"SKYBRU.DAT"としてありますので，このパス名は適当に設定してください．

リスト 6-1 重ね合わせ処理(LIST 6-1.ASM)

```
;
;******* List 6-1 *******
;
VTOP     equ     Ø                      Vram TOP address
HLEN     equ     5ØH                    Horizontal LENgth
NEXTDT   equ     8ØH                    NEXT DaTa
NEXTPT   equ     2ØØH                   NEXT PaTtern
;
GINIT:   ;Graphic system INITialize
         MOV     AX,4ØØØH               グラフィック画面の表示開始コマンドをセットする
         INT     18H                    BIOS コール
         MOV     AX,42ØØH               グラフィック画面モード設定コマンドをセットする
         MOV     CX,ØCØØØH              640×400 ドット・カラー・モードで初期化
         INT     18H                    ROM 内ルーチン・コール
         RET                            リターン
```

```
;
XYADR:  ;XY  to  ADdress           ── (CL,CH)から表示アドレスを求める
        PUSH    AX                 AXレジスタ値をスタックへ退避
        XOR     AX,AX              AXレジスタ初期化
        MOV     AH,CH              Y座標の100H倍を求める
        SHR     AX,1               Y座標の80H倍を求める
        SHL     CH,1               Y座標の200H倍を求める
        ADD     AX,CX              AX←Y×280H+X
        ADD     AX,VTOP            0点座標補正
        MOV     DISPAD,AX          表示アドレス保存
        POP     AX                 AXレジスタ復元
        RET                        リターン
;
KASANE: ;KASANE awase
        PUSH    SI                 ┐
        PUSH    ES                 │レジスタ退避
        PUSH    DS                 ┘
        CALL    XYADR              ┐
        MOV     DI,DISPAD          ┘DI←表示アドレス
        MOV     SI,Ø               SI←グラフィック・データ先頭アドレス
        MOV     AX,GREEN           ┐
        MOV     ES,AX              ┘ES←にG面用セグメント値セット
        MOV     CH,2ØH             Y方向ループ回数
KLP1:
        MOV     CL,2               X方向ループ回数
KLP2:
        MOV     AX,BLUE            ┐
        MOV     DS,AX              ┘DSへB面用セグメント値セット
        MOV     AX,1               ┐
        OUT     ØA6H,AL            ┘裏面アクセスとする
        MOV     DX,[DI]            DX←裏B面のワードデータ
        MOV     BX,[DI+8ØØØH]      BX←裏R面のワードデータ
        MOV     BP,ES:[DI]         BP←裏G面のワードデータ
        XCHG    AL,AH              ┐
        OUT     ØA6H,AL            ┘表面アクセスとする
        MOV     AX,PTNSEG          ┐
        MOV     DS,AX              ┘DSへパターン用セグメント値セット
        MOV     AX,[SI]            AX←透明データ
        AND     DX,AX              ┐
        AND     BX,AX              │透明データと背景とのANDをとる
        AND     BP,AX              ┘
        OR      DX,[SI+NEXTDT]     ┐
        OR      BX,[SI+2*NEXTDT]   │その結果とグラフィックデータとのORをとる
        OR      BP,[SI+3*NEXTDT]   ┘
        MOV     AX,BLUE            ┐
        MOV     DS,AX              ┘DSへB面用セグメント値セット
        MOV     [DI],DX            ┐
        MOV     [DI+8ØØØH],BX      │表示アドレスへデータを入れる
        MOV     ES:[DI],BP         ┘
        ADD     DI,2               表示アドレス更新
        ADD     SI,2               グラフィック・アドレス更新
```

```
        DEC     CL                      } X方向分繰り返す
        JNE     KLP2
        ADD     DI,HLEN-4                 DI←次ラインの表示アドレス
        DEC     CH                      } Y方向分繰り返す
        JNE     KLP1
        POP     DS                      ┐
        POP     ES                      │ レジスタを復元
        POP     SI                      ┘
        RET
;
DISPAD  dw      Ø                         DISPlay ADdress
;
DBACK:  ;Display BACKground               ── 背景の表示
        CALL    XYADR                   ┐
        MOV     CX,BX                   │ BP←表示アドレス
        MOV     BP,DISPAD               ┘
        PUSH    DS
        MOV     AX,BLUE                 } DS←B面用セグメント値
        MOV     DS,AX
        XOR     DX,DX                     DX←0
        XCHG    DL,CH                     DL←X方向ループ回数セット & CH←0
        MOV     AX,GREEN                } ES←G面用セグメント値
        MOV     ES,AX
DBLP1:
        MOV     DH,DL                     DH←X方向ループ回数
        MOV     DI,BP                     DI←表示アドレス
DBLP2:
        MOV     AL,1                    } 裏面アクセスとする
        OUT     ØA6H,AL
        MOV     BL,[DI]                 ┐
        MOV     BH,[DI+8ØØØH]           │ 裏の背景のデータをレジスタへ確保
        MOV     AH,ES:[DI]              ┘
        XOR     AL,AL                   } 表面アクセスとする
        OUT     ØA6H,AL
        MOV     [DI],BL                 ┐
        MOV     [DI+8ØØØH],BH           │ 裏面のデータを表へ表示
        MOV     ES:[DI],AH              ┘
        INC     DI                        表示アドレス更新
        DEC     DH                      } X方向ループ
        JNE     DBLP2
        ADD     BP,HLEN                   BP←次ライン表示
        LOOP    DBLP1                     Y方向ループ
        POP     DS
        RET
;
MVCLS:  ;MoVe CLS
        PUSH    CX                      │
        DEC     AL                      │
        JE      D1CLS                   │
        DEC     AL                      │
        JE      D2CLS                   │
```

```
        DEC     AL
        JE      D3CLS
        DEC     AL
        JE      D4CLS
        DEC     AL
        JNE     $+5
        JMP     D5CLS
        DEC     AL
        JNE     $+5
        JMP     D6CLS
        DEC     AL
        JNE     $+5
        JMP     D7CLS              } 方向別の消去先へジャンプ
;
D8CLS:
        MOV     BX,4Ø8H
        CALL    DBACK
        POP     CX
        INC     CH
        PUSH    CX
        MOV     BX,12ØH
        CALL    DBACK
        POP     CX
        INC     CL
        RET                        } 方向=8の不要部分消去
;
D1CLS:
        MOV     BX,12ØH
        CALL    DBACK
        POP     CX
        PUSH    CX
        ADD     CH,4
        MOV     BX,4Ø8H
        CALL    DBACK
        POP     CX
        INC     CL
        RET                        } 方向=1の不要部分消去
;
D2CLS:
        MOV     BX,12ØH
        CALL    DBACK
        POP     CX
        PUSH    CX
        ADD     CH,3
        MOV     BX,4Ø8H
        CALL    DBACK
        POP     CX
        DEC     CH
        INC     CL
        RET                        } 方向=2の不要部分消去
```

```
;
D3CLS:
        ADD     CH,3
        MOV     BX,4Ø8H
        CALL    DBACK
        POP     CX
        DEC     CH
        RET
;
D4CLS:
        ADD     CH,3
        MOV     BX,4Ø8H
        CALL    DBACK
        POP     CX
        PUSH    CX
        ADD     CL,3
        MOV     BX,12ØH
        CALL    DBACK
        POP     CX
        DEC     CH
        DEC     CL
        RET
;
D5CLS:
        ADD     CL,3
        MOV     BX,12ØH
        CALL    DBACK
        POP     CX
        PUSH    CX
        ADD     CH,4
        MOV     BX,4Ø8H
        CALL    DBACK
        POP     CX
        DEC     CL
        RET
;
D6CLS:
        MOV     BX,4Ø8H
        CALL    DBACK
        POP     CX
        PUSH    CX
        ADD     CL,3
        INC     CH
        MOV     BX,12ØH
        CALL    DBACK
        POP     CX
        INC     CH
        DEC     CL
        RET
```

方向＝3の不要部分消去（D3CLS）

方向＝4の不要部分消去（D4CLS）

方向＝5の不要部分消去（D5CLS）

方向＝6の不要部分消去（D6CLS）

```
;
D7CLS:
        MOV     BX,4Ø8H                 ┐
        CALL    DBACK                   │
        POP     CX                      │ 方向=7の不要部分消去
        INC     CH                      │
        RET                             ┘
;
DATLD:  ;DATa LoaD
        MOV     SI,offset LODTBL        SI←ロード・テーブル・アドレス
DLDLP1: ;Data LoaD LooP 1
        MOV     AL,[SI].CMD             AL←ロードコマンド
        AND     AL,AL                   コマンドエンドか?
        JE      DLD2                    コマンドエンドであればDLD2へ
        CALL    LOAD                    データ・ロード
        ADD     SI,type lodinf          SIレジスタ更新
        JMP     DLDLP1                  DLDLP1へ
DLD2:   ;Data LoaD 2
        RET
;
LOAD:   ;LOAD
        LEA     DX,[SI].PASS            ファイルパス名格納アドレス取得
        MOV     AL,Ø                    ファイル・アクセス・コントロール
        MOV     AH,3DH                  ハンドルのオープン
        INT     21H                     ファンクションコール
        JNB     DOPEN                   エラーがなければDOPENへ
        CMP     AX,2                    ファイルが存在しない場合のチェック
        JNE     DLERR                   エラー・コードによる処理の選択
        LEA     DX,[SI].PASS            ファイルパス名格納アドレス取得
        MOV     AH,9                    ストリングのスクリーン出力
        MOV     [SI].ENDSIN,"$"         ストリング終了コードセット
        INT     21H                     ファンクションコール
        print   EMES2                   エラーメッセージ出力
        MOV     AX,ØFFFFH               エラー・サイン・セット
        JMP     DLRET                   リターン
DLERR:  ;Data Load ERRor
        LEA     DX,[SI].PASS            ファイルパス名格納アドレス取得
        MOV     AH,9                    ストリングのスクリーン出力
        MOV     [SI].ENDSIN,"$"         ストリング終了コードセット
        INT     21H                     ファンクションコール
        print   EMES1                   エラー・メッセージ出力
        MOV     AX,ØFFFFH               エラー・サイン・セット
        JMP     DLRET                   リターン
DOPEN:  ;Data file OPEN
        MOV     HANDL,AX                ハンドル保存
        MOV     BX,AX                   ハンドルを指定レジスタへ
        MOV     CX,Ø                    CX←0
        MOV     DX,CX                   DX←0
        MOV     AH,42H                  ファイル・ポインタの移動とする
        MOV     AL,2                    ポインタをファイルの終わりに移動とする
        INT     21H                     ファイルの大きさをAXレジスタへ
```

```
        MOV     FLLEN,AX                ファイルの大きさを保存
        MOV     BX,HANDL                ハンドルを指定レジスタへ
        MOV     AH,3EH                  ハンドルのクローズ
        INT     21H                     ファンクションコール
        LEA     DX,[SI].PASS            指定ファイルのパス名の格納アドレス取得
        MOV     AL,Ø                    ファイル・アクセス・コントロール
        MOV     AH,3DH                  ハンドルのオープン
        INT     21H                     ファンクションコール
        MOV     HANDL,AX                ハンドルの保存
        MOV     BX,AX                   指定レジスタにハンドルセット
        PUSH    DS                      データ・セグメント値をスタックへ退避
        MOV     DX,[SI].LDADR           ロード・アドレス・セット
        MOV     CX,FLLEN                ファイルの大きさをセット
        MOV     AX,[SI].LDSEG           ロード・セグメント値セット
        MOV     DS,AX                   DSへAXを介してセグメント値セット
        MOV     AH,3FH                  データ・ロード
        INT     21H                     ファンクションコール
        POP     DS                      DSをスタックから復元
        MOV     FILEL,AX                読み込んだバイト数保存
        MOV     BX,HANDL                ハンドルを指定レジスタへ
        MOV     AH,3EH                  ハンドルのクローズ
        INT     21H                     ファンクションコール
        XOR     AX,AX                   ノーマル・リターンとする
DLRET:  ;Data Load RET
        RET                             リターン
;
FILEL   dw      Ø                       ┐
FLLEN   dw      Ø                       │
DISAD   dw      Ø                       ├ ファイル・アクセス用ワークエリア
HANDL   dw      Ø                       ┘
EMES1   db      1Ø,13                   エラー・メッセージ番号1
        db      "ファイル オープン エラー"
        db      1Ø,13,"$"
EMES2   db      1Ø,13                   エラー・メッセージ番号2
        db      "ファイル ガ, アリマセン "
        db      1Ø,13,"$"
CLEAR   db      1BH,"[2J",1BH,"[>1h",1BH,"[>5h$"
CSRON   db      1BH,"[>5l",1BH,"[>1l$"
;
CODE    ends

STACK   segment stack
        db      1ØØH    dup (?)
STACK   ends

MPDSEG  segment                         MaP Data SEGment
        db      ØFFFFH dup (?)
MPDSEG  ends

MPPSEG  segment                         MaP Pattern SEGgment
        db      18ØH*1ØØ dup (?)
MPPSEG  ends
```

```
PTNSEG   segment                              PaTterN SEGgment
         db       2ØØH*1ØØ dup (?)
PTNSEG   ends

MOJSEG   segment                              MOJi SEGgment
         db       8ØH*5Ø  dup (?)
MOJSEG   ends

BLUE     segment  at ØA8ØØH
         db       ØFFFFH dup (?)
BLUE     ends

RED      segment  at ØBØØØH
         db       8ØØØH  dup (?)
RED      ends

GREEN    segment  at ØB8ØØH
         db       8ØØØH  dup (?)
GREEN    ends

         end
```


テスト6-1　テスト・プログラム(TEST 6-1.ASM)

```
;
;****** TEST 6-1 ******
;
CODE     segment                              命令の置かれているセグメントの始まり

         assume CS:CODE,DS:CODE,SS:STACK

print    macro    string                      文字列出力マクロの定義
         LEA      DX,string                   マクロパラメータのオフセットを DX へ
         MOV      AH,9                        ファンクションコール番号 9 …… 文字列出力
         INT      21H                         ファンクションコール
         OUT      64H,AL                      GDC にリセットコマンドを送る
         endm                                 マクロ定義終了

fvram    macro    vramseg,value               VRAM への書き込みマクロ定義
         MOV      DI,Ø                        DI レジスタ初期化
         MOV      AX,vramseg                  AX ← vramseg
         MOV      ES,AX                       ES ← AX
         MOV      AX,value                    AX ← value
         MOV      CX,7DØØH/2                  CX ← 7D00H÷2
         REP      STOSW                       ES：DI ← AX ＆ DI ← DI+2 を CX 回繰り返す
         endm                                 マクロ定義終了
```

```
PTEST:  ;Program TEST
        CLD                              ストリング命令用フラグを＋方向とする
        MOV     AX,CS                    AX へコードセグメント値をセット
        MOV     DS,AX                    DS へコードセグメント値をセット
        print   CLEAR                    テキスト画面クリア
        CALL    GINIT                    グラフィック・システム初期化
        CALL    DATLD                    パターン・データ・ロード
        MOV     AL,1                   } 裏画面面へのアクセス
        OUT     ØA6H,AL                }
        fvram   BLUE,5555H             }
        fvram   RED,Ø                  } 裏の B 面を 5555H で埋める
        fvram   GREEN,Ø                }
        MOV     AL,Ø                   } 表画面面へのアクセス
        OUT     ØA6H,AL                }
        fvram   BLUE,Ø                 }
        fvram   RED,Ø                  } 表の G 面を 5555H で埋める
        fvram   GREEN,5555H            }
        MOV     AX,CS                    AX へコードセグメント値をセット
        MOV     ES,AX                    ES へコードセグメント値をセット
        MOV     CX,1618H                 CX ←表示パターンの初期座標
TINIT:  ;Test INITialize
        MOV     AX,offset DATA           AX ←移動方向のデータのオフセット・アドレス
        MOV     DATAWK,AX                DATAWK の初期化
TLOOP:  ;Test LOOP
        MOV     AX,ØCØØH               } キーボード・バッファ・クリア
        INT     21H                    }
        INC     word ptr DATAWK        }
        MOV     BX,DATAWK              } AL ←次に移動する方向
        MOV     AL,[BX]                }
        OR      AL,AL                  } AL＝0 なら TINIT へ
        JE      TINIT                  }
        CALL    MVCLS                    不要部分の消去
        PUSH    CX
        CALL    KASANE                   重ね合わせ表示
        MOV     CX,8ØØØH               }
WTLP1:                                 } ウェイト処理
        LOOP    WTLP1                  }
        POP     CX
        MOV     AX,Ø4ØØH               }
        INT     18H                    }
        ROR     AH,1                   } ESC が押されていれば MS-DOS へ
        JNB     TLOOP                  }
        MOV     AH,41H                 } グラフィック停止
        INT     18H                    }
        print   CSRON                    カーソル・オン
        MOV     AX,ØCØØH               } キーボード・バッファ・クリア
        INT     21H                    }
        MOV     AL,Ø                   }
        MOV     AH,Ø4CH                } MS-DOS システムへ
        INT     21H                    }
```

```
;
DATAWK  dw      Ø ;DATA Work area
;
QRR     equ     1
QUR     equ     2
QUU     equ     3
QUL     equ     4
QLL     equ     5
QDL     equ     6
QDD     equ     7
QDR     equ     8
;
DATA:   ;direction DATA
        db      QDD,QDD,QDD,QDR
        db      QDD,QDD,QDR,QDD
        db      QDR,QDR,QDR,QRR
        db      QDR,QDR,QRR,QDR
        db      QRR,QRR,QRR,QUR
        db      QRR,QRR,QUR,QRR
        db      QUR,QUR,QUR,QUU
        db      QUR,QUR,QUU,QUR
        db      QUU,QUU,QUU,QUL
        db      QUU,QUU,QUL,QUU
        db      QUL,QUL,QUL,QLL
        db      QUL,QUL,QLL,QUL
        db      QLL,QLL,QLL,QDL
        db      QLL,QLL,QDL,QLL
        db      QDL,QDL,QDL,QDD
        db      QDL,QDL,QDD,QDL
        db      QDD,Ø
;
lodinf  struc
CMD     db      Ø
PASS    db      "           "
ENDSIN  db      Ø
LDADR   dw      Ø
LDSEG   dw      Ø
lodinf  ends
;
LODTBL  lodinf <1,"SKYBRU.DAT" ,Ø,Ø,PTNSEG>
        lodinf < >

        include   LIST6-1.ASM
```

QRR～QDR：方向番号のラベル化

DATA：移動方向を示すデータ

lodinf struc：ロード情報構造体定義
CMD：ロード・コマンド
PASS：パス格納領域（11文字分）
ENDSIN：パス・エンド・コード
LDADR：ロード・アドレス
LDSEG：ロード・セグメント
lodinf ends：構造体定義終了

2. GDCによるスムース・スクロール

スクロールのための章なのに，スクロールに直接関係のない重ね合わせに，かなり時間を費やしてしまいました．しかし，これも自然の成行きでそうなってしまったわけですから，無理に手を抜くこともできなかったのです．本書はもともとこの半分くらいの内容で仕上げる予定だったのですが，書いているうちに『これも，あれも……』ということになり，だんだん中身がふくらんでいってしまったというのが，偽らざる実情なのです．こうなったのも，考えてみると特に誰の責任というのでもなく，やはりこの本自体がそうなるべき運命を最初から持っていたのかもしれません．

さて，PC-9801シリーズには，GDC (Graphic Display Controler)が搭載されています．このGDCには円や直線を描いたり，拡大表示や文字表示などの色々な機能が備わっています．

これらの機能のなかに画面をスクロールさせる機能があります．せっかくある機能ですし，これからスクロールタイプのゲームを作ろうとしているのですから，これを使うのは当然のことです．

とはいえ，このスクロールの仕組みは複雑なため，プログラムはかなりわかりにくいものとなっています．

図6-2を参照してください，GDCを使うとこのように任意のアドレス(ワード単位)で画面を2分割して表示することが可能なのです．

今まで，画面上一番上の一番左側がオフセットアドレスの0となるように設定していましたが，この場合，この0点を分割画面A1の一番上の一番左側に取ります．次に，この0点の位置を細かく連続的に変えていくことによって，画面上はあたかもスクロールしているかのように見せることができるのです．ここでは話を簡単にするためにX方向固定としましたが，もちろんX方向も変化させれば，8方向のスクロールが実現できることになります．0点がズレたのですから，元々の画面上の絵はズレて表示されるわけです．

何か処理が難しそうですが，これを固定座標系から移動座標系に移ったと考えてみ

図6-2 GDCを使った画面分割

0	DATA READY
1	FIFO BUFFER FULL
2	FIFO BUFFER EMPTY
3	DRAWING
4	DMA EXECUTE
5	VERTICAL SYNC
6	HORIZONTAL BLANK
7	LIGHT PEN DETECT

表6-1　GDCのステータス・フラグ

図6-3　GDCのスクロールコマンド

ると，数式上は移動値の単純加算ということになってしまいますから驚くほどのことではありません．

以上のことを踏まえて**リスト6-2**を見てみると，わかりやすいのではないかと思います．

では，次にGDCのコントロールに話を進めることにしましょう．GDCのコントロールは基本的にポートを介して，コマンドとパラメータを送ることで実現できます．コマンド用のポートはA2H，パラメータ用のポートはA0Hです．

また，GDCにはバッファがあり，書き込まれたコマンドやパラメータを，いったんこのバッファに蓄えて，随時読み出して命令を実行していきます．

したがって，このバッファが一杯になっているときにコマンドやパラメータを送っても意味がありませんから，コマンドやパラメータを送り出すときには，このバッファの空き状態をチェックしなければなりません．

GDCには上に示すように8つの状態を示すステータス・フラグがあります．このなかにはFIFO BUFFER EMPTYというフラグがあります．このFIFOはFirst In First Outの略ですが，このフラグがバッファの使用状態を示しています．すなわち，ビット2が立てば(=1)バッファが空いているということです．

次に，GDCに対するスクロールコマンドですが，これには上位ニブルに7，下位ニブルにGDC内部のRAMアドレス(RA)をセットして送り出します(**図6-3**)．このRAは自動的にインクリメント(+1)されますから，連続したRAMアドレスにパラメータを与える場合には，コマンドは1度送るだけで済みます．また，RAの初期値は0です．

コマンドのあとは，必要とされるパラメータを送らなければなりません．**図6-4**のようにA1，A2に2分割して，各分割画面の開始アドレスと，ライン数をそれぞれSAD1，SL1とSAD2，SL2とします．

各パラメータは**表6-2**のような構造として，順番に送り出すように設定されています．

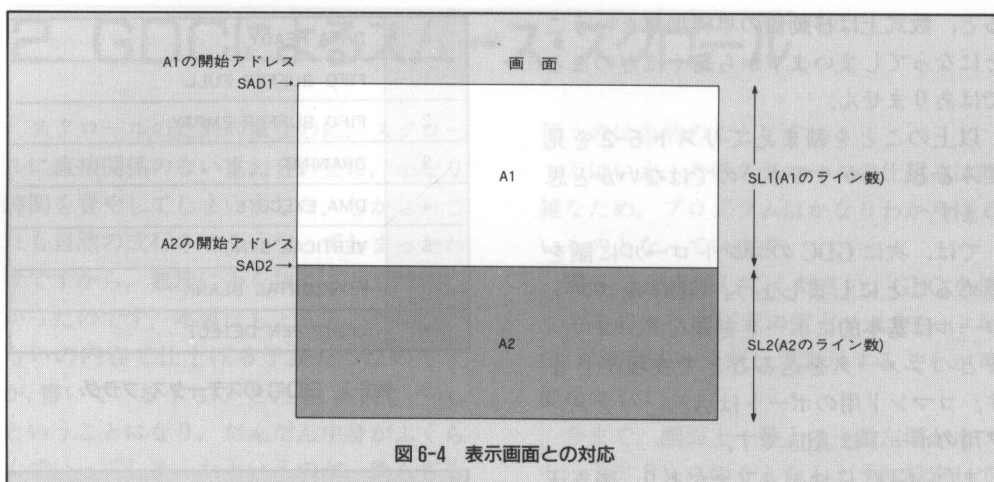



図6-4　表示画面との対応

RA	7	6	5	4	3	2	1	0
0	←			SAD1L				→
1	←			SAD1H				→
2	←	SL1L		→	0	0	0	0
3	＊	IM	←		SL1H			→
4	←			SAD2L				→
5	←			SAD2H				→
6	←	SL2L		→	0	0	0	0
7	＊	IM	←					→

＊：DAD＋2

DAD＋2	機　　能
0	"1"によるインクリメント(DAD＋1→DAD)
1	"2"によるインクリメント(DAD＋2→DAD)

IM	表　示　制　御	L／R　制　御
0	2クロックに1回表示アドレスをインクリメント	CSRFORMコマンドの規定値使用
1	4クロックに1回表示アドレスをインクリメント	L／R＝0に強制

表6-2　スクロールコマンドの各パラメータ

さて，これまでの知識があれば基本的にGDCをコントロールできるということになります．が，事はそれほど単純ではありません．実は，CPUではVRAMがA8000H番地から始まっていましたが，GDCでは4000H番地から始まっているのです．しかも，アクセス単位はワード(2バイト)単位ですから，より複雑になっています．これらのことを考慮したのがリスト6-2というわけです．ここではスクロールを例にとったプログラムですが，残る円や直線を描くといったコマンドに関しても，コマンドとパラメータの設定の仕方が異なるだけですから，ぜひ研究してみてください．

また，GDCの動作クロックについてですが，PC-9801VX以降の機種では，ディップ・スイッチのSW2-8をオンにする(下に下げる)と5MHzに，オフにすると(上に上げる)と2.5MHzになります(スイッチを切り換えた後は再起動しなければならない)．

GDCに対するパラメータもこの動作クロックに合わせる必要があります．たとえば，5MHzの場合にはIMビットが1となり，2.5MHzの場合には0となります．このSW2の状態はポートの31Hを介してチェックすることができますが，スイッチONで〝0″，スイッチOFFで〝1″となっていますから注意してください．

リスト6-2ではこのスイッチの状態を検出してIMビットを立てています．そのあたりも合わせて参考にしてください．

これから作るプログラムは，背景のスクロール，主人公の移動，弾の発射，割り込みによる正確なウェイトとキー入力ですが，目新しいのはGDCによるスクロールと割り込みによるウェイト処理だけで，残りは今までの章でやってきたことと同じです．

まずはウェイトですが，これまでのゲームであれば，たとえ敵がやられて減ったとしても，敵の総数さえ決まっていれば，それに応じた無駄命令で，ある程度の速度調整が可能でした．しかし，スクロールのウェイト処理としてはもっと正確なウェイトが欲しいのです．そこで，リスト6-2ではVSYNC割り込みにより，正確なウェイトをかけています．

CRTは周期的(1秒間に約60回)に画面表示を繰り返しており，VSYNC割り込みは，画面表示が終了するたびに信号が画面トップに戻るときに発生します．

この垂直帰線期間の回数を，割り込みによってカウントし，メイン・ループのなかでつねに一定の回数になるようにしたのが，ここでのウェイトなのです．このVSYNC割り込みで注意しなければならないことは，一度割り込みが発生したらプログラム側でI/Oの64Hに書込みをしなければならないということです．それを怠ると以降この割り込みが発生しなくなってしまいます．

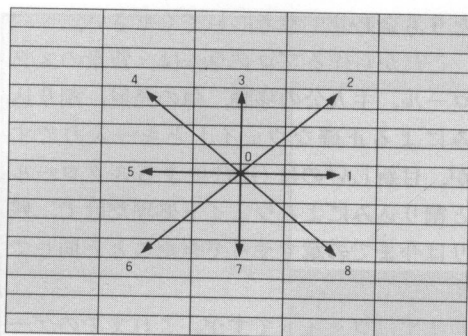



図 6-5 自機の移動方向

さて，これまでゲーム座標ということで，横８ドット，縦８ドットを１コマとしてきました．このスクロールゲームでは，よりなめらかな画面スクロールを実現するために，背景を縦２ドット単位で動かしています．そのために，座標も横８ドット，縦２ドットを１コマとした変則的なものに変えてあります．座標からVRAMに変換するルーチン(XYADR)や，敵との衝突チェック(MYCHK)では，そのあたりを頭に入れてプログラムを見てください．なお，背景がスクロールするということで，厳密には自機の座標を背景と相対して考えなければならないのですが，ここでは無視しています．つまり背景スクロールは単なる視覚上の飾りで，自機が停止(方向＝0)しても背景と連動はせず，２ドット前身(相対的に)するということです．

自機に関しての特徴では，SSKCKルーチンでスペースキーによる弾の連射を可能にさせています．ただし，単に連射を許したのでは，左手はスペースキーを押しっ放し，あるいはキーの上にガムテープを貼る，などというイカサマを平気で考えるのが人間の常というものです．こうなると，弾の発射はキーによらずに，自動的に出すのとまったく同じになってしまいます．そこで，押しっ放しによる連射の場合は，一定の間隔を置くようにして，これを防ぐことにしました．

たかが弾の発射にそれほど気を使うこともない，と思われる方もいるかもしれませんが，市販されているゲームはあらゆる部分にわたって，より細かい配慮がなされていることを忘れないでください．

このリスト6-2でいちばんわかりにくいのは，何といってもスクロールに関するルーチンです．しかし，これがリスト6-2の華ですから，ここを理解しないで先に進むわけにはいきません．ここはひとつ，ジックリと腰を据えてマスターしてください．

リスト6-2a　スクロール処理(LIST 6-2.ASM)

```
;
;******* List 6-2 *******
;
VTOP     equ     Ø                Vram TOP addres
LVTOP    equ     Ø                Land Vram TOP addres
LNDTOP   equ     8Ø*2Ø            LaND TOP addres
LNDEND   equ     7DØØH-8Ø*2       LaND END addres
HLEN     equ     8Ø               Horizontal LENgth
NEXTDT   equ     8ØH              NEXT DaTa
BNEXTDT  equ     6ØH              Bun NEXT DaTa
PTNOFF   equ     Ø                PaTterN OFFset addres
VEND     equ     7DØØH            Vram END
WTIMES   equ     3                Waiting TIMES
YBLNO    equ     2Ø               Y Zahyou BuLlet 番号
LNBASE   equ     Ø
;
VTCT:    ;Vertical Trace CounTer           ── 割り込み処理ウェイト・ルーチン
         PUSH    AX               使用レジスタの退避
         INC     byte ptr CS:COUNT   カウンタの値を+1する
         MOV     AL,2ØH           } EOIの送出
         OUT     Ø,AL             }
         OUT     64H,AL           GDCにリセット・コマンドを送る
         POP     AX               使用レジスタの復元
         IRET                     割り込み処理からのリターン
;
PESC     equ ØØH                  Push ESC  key
PRETK    equ 1CH                  Push RETurn Key
PHCLR    equ Ø3EH                 Push Home CLeaR key
PHELP    equ Ø3FH                 Push HELP key
KHCLR    equ ØBEH                 Kaiho Home CLeaR key
PSPACE   equ 34H                  Push SPACE key1

KSPACE   equ ØB4H                 Kaiho SPACE key
PDOWN    equ 4BH                  Push DOWN key
KDOWN    equ ØCBH                 Kaiho DOWN key
PLEFT    equ 46H                  Push LEFT key
KLEFT    equ ØC6H                 Kaiho LEFT key
PRIGHT   equ 48H                  Push RIGHT key
KRIGHT   equ ØC8H                 Kaiho RIGHT key
PUP      equ 43H                  Push UP key
KUP      equ ØC3H                 Kaiho UP key
;
KEYSET:  ;KEY information SET
         PUSH    AX               }
         PUSH    BX               }
         PUSH    CX               } レジスタの退避
         PUSH    DX               }
         PUSH    DI               }
         PUSH    ES               }
```

```
         CALL    RND                     乱数発生ルーチン・コール
         IN      AL,43H                  ┐
         TEST    AL,38H                  │エラーがあればKEYOUTへ
         JNZ     KEYOUT                  ┘
         MOV     AL,16H                  ┐
         OUT     43H,AL                  │AL←キー情報
         IN      AL,41H                  ┘
         CMP     AL,PESC                 キー情報は[ESC]か
         JNE     NSTOP                   でなければNSTOPへ
         MOV     CS:KPSTOP,1             STOPサインをセット
         JMP     KEYOUT                  KEYOUTへ
NSTOP:   ;Not STOP
         CMP     AL,PRETK                キー情報は[RETURN]か
         JNE     NRETK                   でなければNRETKへ
         MOV     CS:RETKEY,1             RETURNサインをセット
         JMP     KEYOUT                  KEYOUTへ
NRETK:   ;Not RETurn Key
         CMP     AL,PHELP                キー情報は[HELP]キーか
         JNE     NHELP                   でなければNHELPへ
         MOV     CS:KPHELP,1             HELPサインをセット
         JMP     KEYOUT                  KEYOUTへ
NHELP:   ;Not HELP key
         MOV     BX,CS                   BX←CS
         MOV     ES,BX                   ES←BX
         MOV     DI,offset KEYTBL        DI←キー・データ・サーチ用テーブルのオフセット・アドレス
         MOV     CX,12                   キーデータのサーチ回数
         REPNZ   SCASB                   キーデータのサーチ
         JNZ     KEYOUT                  キー情報がサーチ用テーブルになければKEYOUTへ
         MOV     DI,offset KEYINF+12-1   キー情報テーブルのエンド・アドレス
         SUB     DI,CX                   DI←DI−CX
         TEST    AL,80H                  キー情報は解放か？
         JNE     KEYKH                   解放であればKEYKHへ
         MOV     AL,CS:[DI]              AL←キー情報
         OR      CS:KEYDAT,AL            対応するフラグをセット
         JMP     KEYOUT                  KEYOUTへ
KEYKH:   ;KEY KaiHou
         MOV     AL,CS:[DI]              AL←キー情報
         AND     CS:KEYDAT,AL            対応するフラグをリセット
KEYOUT:  ;KEY OUT
         MOV     AL,20H                  ┐
         OUT     0,AL                    ┘EOIの送出
         POP     ES                      ┐
         POP     DI                      │
         POP     DX                      │レジスタ復元
         POP     CX                      │
         POP     BX                      │
         POP     AX                      ┘
         IRET                            割り込み処理からのリターン
```

```
;
KEYDAT  db      Ø                                     KEY DATa
;
KEYTBL  db      PSPACE,PDOWN,PLEFT,PRIGHT,PUP,PHCLR
        db      KSPACE,KDOWN,KLEFT,KRIGHT,KUP,KHCLR
;
KEYINF  db      8ØH,   1H,   2H,   4H,   8H,  1ØH
        db      Ø7FH,ØFEH,ØFDH,ØFBH,ØF7H,ØEFH
;
KPSTOP  db      Ø                                     KeeP STOP
;
KPHELP  db      Ø                                     KeeP HELP
;
RETKEY  db      Ø                                     RETurn KEY
;
COUNT   db      Ø                                     COUNT
;
KPKOFF  dw      Ø                                     KeeP Key OFFset
;
KPKSEG  dw      Ø                                     KeeP Key SEGment
;
KPVTOFF dw      Ø                                     KeeP VrTc OFFset
;
KPVTSEG dw      Ø                                     KeeP VrTc SEGment
;
KPMASK  db      Ø                                     KeeP MASK
;
IREDKEY equ     35Ø9H                                 キーボード割り込みベクタを求めるため
;
IREDVTC equ     35ØAH                                 VSYNC 割り込みベクタを求めるため
;
ISETKEY equ     25Ø9H                                 キーボード割り込みベクタをセットするため
;
ISETVTC equ     25ØAH                                 VSYNC 割り込みベクタをセットするため
;
SETINT: ;SET INTerrupt                                ―― 割り込みモードの設定
        PUSH    DS                                    } レジスタ退避
        PUSH    ES
        XOR     AL,AL                                 } カウンタ値リセット
        MOV     COUNT,AL
        MOV     AX,IREDKEY
        INT     21H                                   } キーボード割り込みベクタを保存
        MOV     KPKOFF,BX
        MOV     KPKSEG,ES
        MOV     AX,ISETKEY
        MOV     DX,offset KEYSET                      } キーボード割り込みベクタを登録する
        INT     21H
        MOV     AX,IREDVTC
        INT     21H                                   } VSYNC 割り込みベクタを保存
        MOV     KPVTOFF,BX
        MOV     KPVTSEG,ES
```

```
        MOV     AX,ISETVTC                       }
        MOV     DX,offset VTCT                   } VSYNC割り込みベクタを登録する
        INT     21H                              }
        CLI                                      }
        IN      AL,2                             }
        MOV     KPMASK,AL                        }
        AND     AL,ØFBH                          } マスク・レジスタ・セット
        OUT     2,AL                             }
        OUT     64H,AL                           }
        STI                                      }
        POP     ES                               } レジスタ復元
        POP     DS                               }
        RET
;
ORIINT: ;ORIginal INTerrupt                      ── 割り込みモードの復元
        PUSH    DS                               }
        LDS     DX,dword ptr CS:KPVTOFF          }
        MOV     AX,ISETVTC                       }
        INT     21H                              } 割り込みベクタを復元
        LDS     DX,dword ptr CS:KPKOFF           }
        MOV     AX,ISETKEY                       }
        INT     21H                              }
        CLI                                      }
        MOV     AL,CS:KPMASK                     }
        OUT     2,AL                             } マスク・レジスタの復元
        STI                                      }
        POP     DS                               }
        RET
;
PWAIT:  ;Program WAIT                            ── ウェイト
        MOV     AL,BEEP
        AND     AL,AL
        JE      W1
        XOR     AL,AL
        MOV     BEEP,AL
        MOV     AL,6                             } ブザー・オン
        OUT     37H,AL                           }
W1:     ;Wait 1
        MOV     AL,COUNT                         }
        CMP     AL,WTIMES                        } カウンタの値が一定の値になるまでループ
        JL      PWAIT                            }
        XOR     AL,AL                            } カウンタの値を0にセット
        MOV     COUNT,AL                         }
        MOV     AL,7                             } ブザー・オフ
        OUT     37H,AL                           }
        RET
```

```
;
BEEP     db       Ø
;
DISP:    ;DISPlay
         PUSH     SI                         レジスタ退避
         PUSH     ES
         CALL     XYADR                      表示アドレスを求める
         CALL     PDADR                      パターン・アドレスを求める
         MOV      SI,BP                      SI←パターン・アドレス
         MOV      DI,DISPAD                  DI←表示アドレス
         PUSH     DS                         レジスタ退避
         MOV      AX,GREEN                   G面セグメント値セット
         MOV      ES,AX
BOX:     ;BOX
         MOV      CH,2ØH                     Y方向ループ値セット
BXLP1:   ;BoX LooP
         MOV      CL,2                       X方向ループ値セット
BXLP2:   ;BoX LooP 1
         MOV      AX,BLUE                    B面セグメント値セット
         MOV      DS,AX
         MOV      AX,1                       裏画面アクセスとする
         OUT      ØA6H,AL
         MOV      DX,[DI]                    各裏画面のデータをレジスタへ
         MOV      BX,[DI+8ØØØH]
         MOV      BP,ES:[DI]
         XCHG     AL,AH                      表画面アクセスとする
         OUT      ØA6H,AL
         MOV      AX,PTNSEG                  DSへPTNSEGをセット
         MOV      DS,AX
         MOV      AX,[SI]                    透明色に対する処理
         AND      DX,AX
         AND      BX,AX
         AND      BP,AX
         OR       DX,[SI+NEXTDT]             パターン・データをセットする
         OR       BX,[SI+2*NEXTDT]
         OR       BP,[SI+3*NEXTDT]
         MOV      AX,BLUE                    DSへBLUEをセット
         MOV      DS,AX
         MOV      [DI],DX                    背景とともにパターンを表示
         MOV      [DI+8ØØØH],BX
         MOV      ES:[DI],BP
         ADD      DI,2                       アドレス更新
         ADD      SI,2
         DEC      CL                         X方向ループ
         JNE      BXLP2
         ADD      DI,HLEN-4                  アドレス更新
         CMP      DI,VEND                    特異点チェック
         JB       BLØØ3                      特異点でなければXYAD1へ
         SUB      DI,VEND                    特異点補正
BLØØ3:   ;Box Label ØØ3
         DEC      CH                         Y方向ループ
         JNE      BXLP1
```

```
        POP     DS
        POP     ES                      } レジスタ復元
        POP     SI
        RET
;
DISPAD  dw      Ø                         DISPlay ADdress
;
XYADR:  ;XY to ADdress                    ──(CL, CH)から表示アドレスを求める
        PUSH    AX                        AXレジスタ値をスタックへ退避
        XOR     AX,AX                     AXレジスタ初期化
        MOV     AH,CH                     Y座標の100H倍を求める
        SHR     AX,1                      Y座標の80H倍を求める
        SHR     AX,1                        〃  40H  〃
        ADD     AH,CH                       〃 140H  〃
        SHR     AX,1                        〃 0A0H  〃
        MOV     CH,Ø                      CH←0
        ADD     AX,CX                     AX←Y×160+X
        ADD     AX,VTOP                   0点座標補正
CONECT: ;CONECT
        ADD     AX,IDOZAH                 移動値補正
        JS      HOSEI                     負であればHOSEIへ
        CMP     AX,VEND                   特異点チェック
        JB      XYAD1                     特異点でなければXYAD1へ
HOSEI:  ;HOSEI
        SUB     AX,VEND                   特異点補正
XYAD1:  ;XY to ADress 1
        MOV     DISPAD,AX                 表示アドレス保存
        POP     AX                        AXレジスタ復元
        RET                               リターン

TOPADR: ;TOP ADdRess
        PUSH    AX                      } トップ・アドレスを求める
        MOV     AX,LVTOP
        JMP     CONECT
LNDADR: ;LaND ADdRess
        PUSH    AX                      } 背景トップ・アドレスを求める
        MOV     AX,LNDTOP
        JMP     CONECT
LDEADR: ;LanD End ADdRess
        PUSH    AX                        背景エンド・アドレスを求める
        MOV     AX,LNDEND
        JMP     CONECT
;
PDADR:  ;Pattern Data ADdRess             ──データ・アドレスをBPレジスタへ求める
        MOV     AH,Ø                      AXの上位レジスタを初期化
        XCHG    AL,AH                     AX←AL×100H
        SHL     AX,1                      AX←AX×2 …… AX←AL×200H
        MOV     BP,AX                     データ格納アドレスをBPへ
        RET                               リターン
;
CLPTXY: ;CLear PaTtern  (X, Y)
        MOV     BP,BX                     BP←BX
```

```
        CALL    XYADR           表示アドレスを求める
        MOV     CX,BP           CX←BP
        MOV     BP,DISPAD       BP←表示アドレス
        PUSH    DS              レジスタ保存
        MOV     AX,BLUE         DSへBLUEをセット
        MOV     DS,AX
        XOR     DX,DX           DX←0
        XCHG    DL,CH           DL←CH
        MOV     AX,GREEN        ESへGREENをセット
        MOV     ES,AX
EBLP1:  ;Erase Box LooP 1
        MOV     DH,DL           X方向ループ値セット
        MOV     DI,BP           表示アドレス・セット
EBLP2:  ;Erase Box LooP 2
        MOV     AL,1            }裏画面アクセスとする
        OUT     ØA6H,AL
        MOV     BL,[DI]         }
        MOV     BH,[DI+8ØØØH]   }裏画面背景データをレジスタへ
        MOV     AH,ES:[DI]      }
        XOR     AL,AL           }表画面アクセスとする
        OUT     ØA6H,AL
        MOV     [DI],BL         }
        MOV     [DI+8ØØØH],BH   }背景データを復元する
        MOV     ES:[DI],AH      }
        INC     DI              アドレス更新
        DEC     DH              }Y方向ループ
        JNE     EBLP2
        ADD     BP,HLEN         表示アドレス更新
        CMP     BP,VEND         }
        JB      NOTVEN2         }特異点補正
        SUB     BP,VEND         }
NOTVEN2:;NOT V-ram ENd 2
        LOOP    EBLP1           Y方向ループ
        POP     DS              レジスタ復元
        RET                     リターン
;
blwrite macro   reg             弾表示用マクロ定義
        local   BLNVEØ
        OR      [DI],reg
        OR      [DI+8ØØØH],reg
        OR      ES:[DI],reg
        ADD     DI,HLEN
        OR      [DI],reg
        OR      [DI+8ØØØH],reg  各VRAMへregの内容を2ライン分書き込む
        OR      ES:[DI],reg
        ADD     DI,HLEN
        CMP     DI,VEND
        JB      BLNVEØ
        SUB     DI,VEND
BLNVEØ:
        endm
```

```
;
BLPUT:  ;BuLlet Put
        PUSH    DS                  ── (CL, CH)に弾を表示
                                    レジスタ保存
        CALL    XYADR               表示アドレスを求める
        MOV     DI,DISPAD           DI←表示アドレス
        MOV     AX,BLUE
        MOV     DS,AX
        MOV     AX,GREEN            VRAMセグメント・セット
        MOV     ES,AX
        MOV     AL,7EH
        blwrite AL
        MOV     AL,ØFFH
        blwrite AL                  弾の表示
        MOV     AL,7EH
        blwrite AL
        POP     DS                  レジスタ復元
        RET                         リターン
;
DISPLE: ;DISPlay LEtter             ── 文字・数字の表示
        PUSH    SI                  SIレジスタ値をスタックへ退避
        PUSH    DS                  DSレジスタ値をスタックへ退避
        CALL    XYADR               表示アドレスを求めるため
        CALL    SEEKLD              データのアドレスを求めるため
        MOV     SI,AX               SI, AX
BOXL:   ;BOX of Letter              ── 1文字の表示
        MOV     DI,DISPAD           DI←表示アドレス
        MOV     CX,16               CX←縦のドット数
        MOV     AX,MOJSEG           文字列パターンの格納されたセグメント・アドレス
        MOV     DS,AX               AXレジスタを介してDSにセグメント値を格納
LLOOP:  ;Letter LOOP
        MOV     AX,BLUE             AX←B面セグメント値
        MOV     ES,AX               AXを介してESにセグメント値を格納
        MOV     DX,[SI]             DX←DS:[SI]
        AND     ES:[DI],DX          ES:[DI]←ES:[DI] AND DX
        MOV     AX,[SI+2ØH]         AX←DS:[SI+20H]
        OR      ES:[DI],AX          ES:[DI]←ES:[DI] OR AX
        MOV     AX,RED              AX←R面セグメント値
        MOV     ES,AX               AXを介してESにセグメント値を格納
        AND     ES:[DI],DX          ES:[DI]←ES:[DI] AND DX
        MOV     AX,[SI+4ØH]         AX←DS:[SI+40H]
        OR      ES:[DI],AX          ES:[DI]←ES:[DI] OR AX
        MOV     AX,GREEN            AX←G面セグメント値
        MOV     ES,AX               AXを介してESにセグメント値を格納
        AND     ES:[DI],DX          ES:[DI]←ES:[DI] AND DX
        MOV     AX,[SI+6ØH]         AX←DS:[SI+60H]
        OR      ES:[DI],AX          ES:[DI]←ES:[DI] OR AX
        ADD     DI,HLEN             表示アドレスを次ラインにする
        CMP     DI,VEND             特異点チェック
        JB      BOXØ2               特異点でなければXYAD1へ
        SUB     DI,VEND             特異点補正
```

```
BOXØ2:  ;BOX Ø2
        ADD     SI,2            SI←SI+2
        LOOP    LLOOP           CX回ループ
        POP     DS              スタックからDSレジスタ値を復元
        POP     SI              スタックからSIレジスタ値を復元
        RET                     リターン
;
SEEKLD: ;SEEK Letter Data
        MOV     AH,Ø            AH←0
        XCHG    AL,AH           AX←文字サーチ・コード番号×100H
        SHR     AX,1            AX←AX÷2 …… コード番号×80Hに相当
        RET                     リターン
;
cstosw  macro   vseg            CLEAR用マクロ定義
        MOV     AX,vseg
        MOV     ES,AX
        MOV     CX,4Ø
        MOV     DI,BP
        XOR     AX,AX
        REP     STOSW
        endm
;
CLS:    ;CLear Screen
        CALL    TOPADR          VRAMトップ・アドレスを求める
        MOV     DX,4ØØ          DXにY方向ループ数セット
ACLS:   ;All CLear Screen
        MOV     BP,DISPAD       BPトップ・アドレス
        XOR     AX,AX           AX←0
CLSLP1: ;CLear Screen LooP 1
        cstosw  BLUE            ┐
        cstosw  RED             │
        cstosw  GREEN           │
        MOV     AL,1            │
        OUT     ØA6H,AL         │
        XOR     AX,AX           ├ 各VRAMに0を書き込む
        cstosw  BLUE            │
        cstosw  RED             │
        cstosw  GREEN           │
        OUT     ØA6H,AL         ┘
        MOV     BP,DI           ┐
        CMP     DI,VEND         │
        JB      CLSVEND         ├ アドレス更新
        SUB     BP,VEND         ┘
CLSVEND:
        DEC     DX              ┐
        JNE     CLSLP1          ┘ 400ライン分ループ
        RET                     リターン
```

```
;
MSGPRN: ;MeSsaGe PRiNt
        MOV     AL,CS:[BX]
        OR      AL,AL
        JNE     $+3
        RET
        CMP     AL,' '
        JNE     MSG2
        MOV     AL,'Ø'+1Ø
MSG2:   ;MSGprn 2
        SUB     AL,'Ø'
        CMP     AL,11
        JB      MSG1
        SUB     AL,6
MSG1:   ;MSGprn 1
        PUSH    CX
        PUSH    BX
        CALL    DISPLE
        POP     BX
        POP     CX
        INC     CL
        INC     CL
        INC     BX
        JMP     MSGPRN
;
SCLOC   equ     Ø3BH*2
;
DISPSC: ;DISPlay SCore
        MOV     BX,offset SCOREL
        MOV     AL,DL
        ADD     AL,CS:[BX]
        DAA
        MOV     CS:[BX],AL
        MOV     AL,DH
        ADC     AL,CS:[BX-1]
        DAA
        MOV     CS:[BX-1],AL
        MOV     AL,Ø
        ADC     AL,CS:[BX-2]
        DAA
        MOV     CS:[BX-2],AL
        SUB     BX,2
        MOV     DI,SCLOC
        CALL    DISPSØ
        RET

DISPSØ: ;DISPlay SCore
        MOV     PRTON,Ø
        CALL    SCOREP
        INC     BX
        CALL    SCOREP
```

(CL, CH)より[BX]で示される
文字列を表示する　*リスト3-2参照

SCore LOCation

```
        INC     BX
        CALL    SCOREP
        MOV     AL,PRTON
        AND     AL,AL
        JE      DSPCRT
        PUSH    DS
        MOV     AX,ØAØØØH
        MOV     DS,AX
        MOV     [DI],byte ptr 3ØH
        MOV     [DI+2],byte ptr 3ØH
        POP     DS
DSPCRT:
        RET
                                        現在のスコアに DX で示される
SCOREP:                                 得点を加算して表示する
        MOV     DL,CS:[BX]
        SHR     DL,1
        SHR     DL,1
        SHR     DL,1
        SHR     DL,1
        CALL    PRINTF
        MOV     DL,CS:[BX]
;
PRINTF:
        MOV     AL,PRTON
        AND     AL,AL
        JNE     PRINT1
        AND     DL,ØFH
        JE      PRRET
        MOV     PRTON,1
        JMP     PRINT2
PRINT1:
        AND     DL,ØFH
PRINT2:
        PUSH    DS
        OR      DL,3ØH
        MOV     AX,ØAØØØH
        MOV     DS,AX
        MOV     [DI],DL
        POP     DS
        ADD     DI,2
PRRET:
        RET
;
PRTON   db      Ø                       PRinT ON
;
SCORE2  db      Ø                       SCORE 2
;
SCORE1  db      Ø                       SCORE 1
;
SCOREL  db      Ø                       SCORE Low
```

```
;
RND:     ;RaNDom figure                            ── ALに乱数を求める
         PUSH     BX
         MOV      BX,word ptr CS:[RNDWOK]
         MOV      DX,BX
         ADD      BX,BX
         ADD      BX,BX
         ADD      BX,DX
         MOV      DX,3573H
         ADD      BX,DX
         MOV      word ptr CS:[RNDWOK],BX
         MOV      AL,BH
         POP      BX
         RET
;
RNDWOK   db       113,31                           RaNDom figure WOrK area
;
MYMOVE: ;MY MOVement
         MOV      CX,MYLOC                         CX←自機の(X,Y)座標
         MOV      AL,KEYDAT                        AL←キー情報
         AND      AL,ØFH                           AL←方向データのセレクト
         SHL      AL,1                             AL←AL×2
         MOV      AH,Ø                           ┐
         MOV      BX,offset DIRTBL               │
         ADD      BX,AX                          ├ 方向別の処理アドレスを得る
         MOV      AX,[BX]                        ┘
         JMP      AX                               方向別の処理へ

         INCLUDE LIST6-2B.ASM
```

リスト6-2b スクロール処理(LIST 6-2 B.ASM)

```
DIRTBL  dw       offset KSTOP,offset KEY2 ,offset KEY4 ,offset KEY24
        dw       offset KEY6 ,offset KEY26,offset KSTOP,offset KEY2
        dw       offset KEY8 ,offset KSTOP,offset KEY48,offset KEY4
        dw       offset KEY68,offset KEY6 ,offset KEY8 ,offset KSTOP
        dw       offset KSTOP
;
KEY24:  ;pressed KEY = 2+4
        CMP      CH,DEND - 6           } CH=DEND(下エンド-6)ならKEY4へ
        JB       DEND24
        JMP      KEY4
DEND24:
        CMP      CL,Ø                  } CL=0ならK2OKへ
        JG       K24OK
        JMP      K2OK
K24OK:  ;Key 2+4 direction OK
        MOV      BX,4Ø8H               } 移動方向=6の不要部分消去+次座標計算
        CALL     CLPTXY
        MOV      CX,MYLOC
        ADD      CL,3
        ADD      CH,4
        MOV      BX,118H+SCLDOT
        CALL     CLPTXY
        MOV      CX,MYLOC
        ADD      CH,4
        DEC      CL
        JMP      MMDISP                  MMDISPへ
;
KEY2:   ;pressed KEY = 2
        CMP      CH,DEND - 6           } CH=DEND(下エンド-6)ならKSTOPへ
        JB       K2OK
        JMP      KSTOP
K2OK:   ;Key 2 direction OK
        MOV      BX,4Ø8H               } 移動方向=7の不要部分消去+次座標計算
        CALL     CLPTXY
        MOV      CX,MYLOC
        ADD      CH,4
        JMP      MMDISP                  MMDISPへ
;
KEY26:  ;pressed KEY = 2 + 6
        CMP      CL,REND               } CL=REND(右エンド)ならKEY2へ
        JL       NRE26
        JMP      KEY2
NRE26:  ;Not Right End 26
        CMP      CH,DEND - 6           } CH=DEND(下エンド-6)ならK6OKへ
        JB       K26OK
        JMP      K6OK
```

```
K26OK:  ;Key 2+6 direction OK
        MOV     BX,4Ø8H
        CALL    CLPTXY
        MOV     CX,MYLOC
        ADD     CH,4
        PUSH    CX
        MOV     BX,118H+SCLDOT
        CALL    CLPTXY
        POP     CX
        INC     CL
        JMP     MMDISP
;
KEY4:   ;pressed KEY = 4
        CMP     CL,Ø
        JG      K4OK
        JMP     KSTOP
K4OK:   ;Key 4 direction OK
        ADD     CL,3
        MOV     BX,12ØH
        CALL    CLPTXY
        MOV     CX,MYLOC
        ADD     CH,16
        MOV     BX,4Ø2H
        CALL    CLPTXY
        MOV     CX,MYLOC
        DEC     CL
        JMP     MMDISP
;
KEY6:   ;pressed KEY = 6
        CMP     CL,REND
        JL      K6OK
        JMP     KSTOP
K6OK:   ;Key 6 direction OK
        MOV     BX,12ØH
        CALL    CLPTXY
        MOV     CX,MYLOC
        ADD     CH,16
        MOV     BX,4Ø2H
        CALL    CLPTXY
        MOV     CX,MYLOC
        INC     CL
        JMP     MMDISP
;
KEY48:  ;pressed KEY = 4+8
        CMP     CL,Ø
        JG      K48N8
        JMP     KEY8
K48N8:  ;K48 Not 8
        CMP     CH,32
        JNB     K48OK
        JMP     K4OK
```

移動方向=8の不要部分消去+次座標計算

MMDISPへ

CL=0(左エンド)ならKSTOPへ

移動方向=5の不要部分消去+次座標計算

MMDISPへ

CL=REND(右エンド)ならKSTOPへ

移動方向=1の不要部分消去+次座標計算

MMDISPへ

CL=0(左エンド)ならKEY8へ

CH=32(上エンド)ならK4OKへ

```
K48OK:   ;Key 4+8 direction OK
         ADD       CL,3
         MOV       BX,118H
         CALL      CLPTXY
         MOV       CX,MYLOC
         ADD       CH,12                      移動方向=4の不要部分消去+次座標計算
         MOV       BX,4Ø8H+SCLDOT
         CALL      CLPTXY
         MOV       CX,MYLOC
         SUB       CH,4
         DEC       CL
         JMP       MMDISP                     MMDISPへ
;
KEY8:    ;pressed KEY = 8
         CMP       CH,32                      CH=32(上エンド)ならKSTOPへ
         JNB       K8OK
         JMP       KSTOP
K8OK:    ;Key 8 direction OK
         ADD       CH,12
         MOV       BX,4Ø8H+SCLDOT             移動方向=3の不要部分消去+次座標計算
         CALL      CLPTXY
         MOV       CX,MYLOC
         SUB       CH,4
         JMP       MMDISP                     MMDISPへ
;
KEY68:   ;pressed KEY = 6+8
         CMP       CH,32                      CH=32(上エンド)ならKEY6へ
         JNB       K68N6
         JMP       KEY6
K68N6:   ;K68 Not key 6
         CMP       CL,REND                    CL=REND(右エンド)ならK8OKへ
         JL        K68OK
         JMP       K8OK
K68OK:   ;Key 6+8 direction OK
         MOV       BX,118H
         CALL      CLPTXY
         MOV       CX,MYLOC
         ADD       CH,12                      移動方向=3の不要部分消去+次座標計算
         MOV       BX,4Ø8H+SCLDOT
         CALL      CLPTXY
         MOV       CX,MYLOC
         SUB       CH,4
         INC       CL
         JMP       MMDISP                     MMDISPへ
;
KSTOP:   ;Key STOP
         ADD       CH,16
         MOV       BX,4ØØH+SCLDOT             移動方向=0の不要部分消去
         CALL      CLPTXY
         MOV       CX,MYLOC
         JMP       MDISP                      MDISPへ
```

```
MMDISP: ;My Move & DISPlay
        MOV     MYLOC,CX                    MYLOC ← CX
MDISP:  ;My DISPlay
        XOR     AL,AL                       (CL, CH)に自機(パターン番号0)を表示する
        CALL    DISP
        RET
;
MYCHK:  ;MY CHeck                           ── 自機と敵の弾, 敵との衝突チェック
        MOV     DX,MYLOC                    DX ←自機の(X, Y)座標
        MOV     BX,offset EMBWOK            BX ←敵の弾のワークエリアの先頭アドレス
        MOV     CX,EMBVAL                   CX ←敵の弾の総数
EMBCLP: ;EneMy Bullet Check LooP
        MOV     AL,[BX].BL_STATUS
        OR      AL,AL                       弾が出現していなければ IBX2 へ
        JE      IBX2
        MOV     AL,[BX].BL_XZAHYOU
        SUB     AL,DL                       敵の弾の X 座標－自機の X 座標≧4 なら IBX2 へ
        CMP     AL,4
        JNB     IBX2
        MOV     AL,[BX].BL_YZAHYOU
        SUB     AL,DH                       敵の弾の Y 座標－自機の Y 座標≧14 なら IBX2 へ
        CMP     AL,14
        JNB     IBX2
        JMP     MYCRT                       MYCRT へ
IBX2:   ;Inc BX 2
        ADD     BX,type bl_info             ワークエリア・アドレス更新
        LOOP    EMBCLP                      弾の数だけループする
;
        MOV     BX,offset EMWORK            BX ←敵のワークエリア先頭アドレス
        MOV     CX,EMVAL                    CX ←敵の総数
WECLP:  ;With Enemy Check LooP
        MOV     AL,[BX].STATUS              敵が出現中(FFH)でなければ NEXTE へ
        INC     AL
        JNE     NEXTE
        MOV     AL,[BX].PATTERN             敵パターン番号<SKYP1(=4)すなわち,
        CMP     AL,SKYP1                    地上敵の場合は NEXTE へ
        JB      NEXTE

        MOV     AL,[BX].XZAHYOU
        SUB     AL,DL
        ADD     AL,4                        敵の X 座標－自機の X 座標+4≧8 なら NEXTE へ
        CMP     AL,8
        JNB     NEXTE

        MOV     AL,[BX].YZAHYOU
        SUB     AL,DH
        ADD     AL,16                       敵の Y 座標－自機の Y 座標+16≧32
        CMP     AL,32
        JB      MYCRT
```

```
NEXTE:  ;NEXT Enemy
        ADD     BX,type eminfo          ワークエリア・アドレス更新
        LOOP    WECLP
        OR      AL,AL
        RET
MYCRT:
        MOV     AL,HELP
        SHR     AL,1
        RET
;
HELP    db      1
;
SSKCK:  ;Set data & Space Key Check
        MOV     BX,offset SSKEY
        MOV     AL,KEYDAT
        AND     AL,80H
        JNE     SSP
        MOV     byte ptr [BX],0FFH
        RET
SSP:    ;Set SPace
        INC     byte ptr [BX]           [BX]←[BX]+1
        JE      $+3                     [BX]≠0ならリターン
        RET
        MOV     byte ptr [BX],0ECH      [BX]← 0ECH
        MOV     CH,MYBVAL               CH ← MYBVAL
        MOV     BX,offset MYBWOK        BX ←自機の弾のワークエリア先頭アドレス
        MOV     CL,0                    CL ← 0 …… 左側の弾(自機のX座標との差)
        CALL    BWCK                    弾の発射準備
        JNB     $+3                     ワークエリアに空きがなければリターン
        RET
        DEC     CH                      CH ← CH−1
        JNE     $+3                     CH=0ならリターン
        RET
        MOV     CL,3                    CL ← 3 …… 右側の弾(自機のX座標との差)
BWCK:   ;Bullet Work area Check
        MOV     AL,[BX].MB_STATUS
        OR      AL,AL
        JE      NBOK
        ADD     BX,type mb_info         弾のワークエリアに空きがあればNBOKへ
        DEC     CH                      なければキャリー・フラグをセットしてリターン
        JNE     BWCK
        STC
        RET
NBOK:   ;New Bullet Ok
        MOV     [BX].MB_STATUS,1
        MOV     AX,MYLOC
        ADD     AL,CL
        MOV     [BX].MB_XZAHYOU,AL      弾出現のフラグ,座標の設定
        MOV     [BX].MB_YZAHYOU,AH
        ADD     BX,type mb_info
        RET
```

```
;
MYBMOV: ;MY Bullet MOVe
        MOV     BX,offset MYBWOK            BX←自機の弾のワークエリア先頭アドレス
        MOV     CX,MYBVAL                   CX←自機の弾の総数
MYBLP:  ;MY Bullet LooP
        MOV     AL,[BX].MB_STATUS         ┐
        OR      AL,AL                     │ 出現中でなければ NEXTMB へ
        JE      NEXTMB                    ┘
        PUSH    CX                        ┐
        MOV     CL,[BX].MB_XZAHYOU        │
        MOV     CH,[BX].MB_YZAHYOU        │ CL←自機の弾の X 座標
        PUSH    BX                        │ CH←自機の弾の Y 座標
        MOV     BX,108H+SCLDOT            │ (CL,CH)にある弾の消去をする
        CALL    CLPTXY                    │
        POP     BX                        ┘
        MOV     CL,[BX].MB_XZAHYOU        ┐
        MOV     CH,[BX].MB_YZAHYOU        │
        SUB     CH,4                      │ CH−4>8 ならば MBPUT へ
        CMP     CH,8                      │
        JNB     MBPUT                     ┘
        MOV     [BX].MB_STATUS,0          ┐
        POP     CX                        │ 自機の出現フラグをリセットし,NEXTMB へ
        JMP     NEXTMB                    ┘
MBPUT:  ;MY Bullet PUT
        MOV     [BX].MB_YZAHYOU,CH        ┐
        PUSH    BX                        │
        CALL    BLPUT                     │ (CL,CH)に弾の表示
        POP     BX                        │
        POP     CX                        ┘
NEXTMB: ;NEXT My Bullet
        ADD     BX,type mb_info             ワークエリア・アドレスの更新
        LOOP    MYBLP                       弾数分ループ
        RET                                 リターン
;
GINIT:  ;Graphic system INITialize
        MOV     AX,4000H                    グラフィック画面の表示開始コマンドをセットする
        INT     18H                         BIOS コール
        MOV     AX,4200H                    グラフィック画面モード設定コマンドをセットする
        MOV     CX,0C000H                   640×400 ドット・カラー・モードで初期化
        INT     18H                         ROM 内ルーチン・コール
        IN      AL,31H                      AL←SW8
        TEST    AL,80H                      GDC モードテスト
        JNE     GIRET                       2.5MHz であれば GIRET へ
        MOV     CS:GDCMD,40H                GDC モードセット
GIRET:  RET                                 リターン
;
SCLDOT  equ     2                           SCroLl DOT number
SCLPTC  equ     SCLDOT*40                   SCroLl PiTCh
IDOZAH  dw      SCLPTC*2                    IDOu ZAHyou
```

```
;
gdcout  macro    reg
        ifidn    <reg>,<AH>
         XCHG     AL,AH
        endif
        OUT      ØAØH,AL
        NOP
        NOP
        endm

SCROLL: ;SCROLL
        CALL     VWAIT
        MOV      AX,IDOZAH
        SUB      AX,SCLPTC*2
        CMP      AX,Ø
        JG       STORE
        MOV      AX,VEND
STORE:  ;STORE
        MOV      IDOZAH,AX
FIFOCK: ;FIFO Check
        IN       AL,ØAØH
        TEST     AL,4H
        JZ       FIFOCK
        MOV      AL,7ØH
        OUT      ØA2H,AL
        MOV      SI,SCLAD1
        SUB      SI,SCLPTC
        MOV      SCLAD1,SI
        MOV      CX,SCLDOT
        CMP      SI,4ØØØH
        JG       CALDAT
        XOR      AX,AX
        MOV      SCLNO1,AX
        MOV      DI,4ØØØH+3E8ØH
        MOV      SCLAD1,DI
        SUB      DI,SCLPTC
        MOV      BX,19ØH
        MOV      SCLNO2,BX
        SUB      BX,CX
        MOV      DX,CX
        JMP      START
CALDAT: ;CALculator DATa
        MOV      BX,SCLNO1
        ADD      BX,CX
        MOV      SCLNO1,BX
        MOV      DX,SCLNO2
        SUB      DX,CX
        MOV      SCLNO2,DX
        MOV      DI,4ØØØH
```

GDCへのパラメータ出力用マクロ定義

ウェイト

移動座標の更新

GDCバッファの空きチェック

スクロール・コマンドの送出

各パラメータの更新

```
START:  ;START
        MOV     CX,4
        MOV     AX,SI
        gdcout  AL
        gdcout  AH
        MOV     AX,BX
        SHL     AX,CL
        gdcout  AL
        OR      AH,GDCMD
        gdcout  AH                  GDCへパラメータを送出
        MOV     AX,DI
        gdcout  AL
        gdcout  AH
        MOV     AX,DX
        SHL     AX,CL
        gdcout  AL
        OR      AH,GDCMD
        gdcout  AH
        CALL    DISMAP              背景の表示
        RET                         リターン
;
VWAIT:  IN      AL,ØAØH             get vertical trace
        TEST    AL,ØØ1ØØØØØB        1=vertical trace
        JNE     VWAIT
VWAT2:  IN      AL,ØAØH
        TEST    AL,ØØ1ØØØØØB
        JE      VWAT2
        RET
;
GDCMD   DB      Ø                   GDCモード
;
DISMAP: ;DISplay MAP
        CALL    LDEADR              背景エンド・アドレスを求める
        MOV     BP,DISPAD           BP←背景エンド・アドレス
        MOV     AX,BLUE
        MOV     ES,AX
        MOV     CX,8Ø
        XOR     AX,AX
        MOV     DI,BP
        REP     STOSW
        MOV     AX,RED
        MOV     ES,AX
        MOV     CX,8Ø               背景不要部分の消去
        XOR     AX,AX
        MOV     DI,BP
        REP     STOSW
        MOV     AX,GREEN
        MOV     ES,AX
        MOV     CX,8Ø
        XOR     AX,AX
        MOV     DI,BP
        REP     STOSW
```

```
         MOV     BX,MAPADR           BX ←マップ・アドレス
         CALL    LNDADR              背景トップ・アドレスを求める
         MOV     BP,DISPAD           BP ←背景トップ・アドレス
         PUSH    DS                  レジスタ退避
         MOV     CX,2Ø               X 方向ループ数のセット
DISML1: ;DISplay Map Loop 1
         MOV     AX,MPDSEG         } マップ・データ用セグメントのセット
         MOV     DS,AX
         MOV     AH,[BX]           }
         CMP     AH,ØFFH           } マップ・データが終わりでなければ DISM02 へ
         JNE     DISMØ2
         MOV     CS:MAPEND,ØFFH      マップ終了サインをセット
         MOV     AH,CS:KPMAPN        表示マップ・パターンは前回のものとする
         JMP     DISMØ3              DISM03 へ
DISMØ2: ;DISplay Map Ø2
         MOV     DX,CS:MPBADR        DX ←マップ・パターン・ベース・アドレス
         AND     DX,DX               DX は 0 か？
         JNE     NTTKID              0 でなければ NTTKID へ
         TEST    AH,8ØH              マップ・データに敵が存在するか？
         JE      NTTKID              敵が存在しなければ NTTKID へ
         MOV     AL,[BX+8ØØØH]       AL ←敵のパターン番号
         MOV     CS:EMYNO,AL         CS：EMYNO ← AL
         MOV     DX,BP             }
         SUB     DX,CS:DISPAD      } 敵の初期 X 座標を求める
         MOV     CS:ENEMYX,DL      }
         ifdef   EMAPP             } 敵起動ルーチン EMAPP が定義
           CALL    EMAPP           } されていればコールする
         endif
NTTKID: ;NoT TeKI Display
         AND     AH,7FH            } 最上位ビットをクリアし
         MOV     CS:KPMAPN,AH      } パターン番号を保存
DISMØ3: ;DISplay Map Ø3
         MOV     AL,Ø                AX ← 100H×パターン番号
         MOV     SI,AX               SI ← AX
         SHR     AX,1                AX ← AX÷2 …… 80H×パターン番号
         ADD     SI,AX               SI ← 180H×パターン番号
         ADD     SI,CS:MPBADR        SI ←マップ・パターン・データ
         MOV     DX,SI
         MOV     AX,MPPSEG
         MOV     DS,AX
         MOV     DI,BP
         MOV     AX,BLUE
         MOV     ES,AX
         MOVSW
         MOVSW
         ADD     DI,8Ø-4
         MOVSW
         MOVSW
         ADD     SI,NEXTDT-8
         MOV     DI,BP
```

```
MOV     AX,RED
MOV     ES,AX
MOVSW
MOVSW
ADD     DI,80-4
MOVSW
MOVSW
ADD     SI,NEXTDT-8
MOV     DI,BP
MOV     AX,GREEN
MOV     ES,AX
MOVSW
MOVSW
ADD     DI,80-4
MOVSW
MOVSW
MOV     AL,1
OUT     0A6H,AL
MOV     SI,DX
MOV     DI,BP
MOV     AX,BLUE
MOV     ES,AX
MOVSW
MOVSW
ADD     DI,80-4
MOVSW
MOVSW
ADD     SI,NEXTDT-8
MOV     DI,BP
MOV     AX,RED
MOV     ES,AX
MOVSW
MOVSW
ADD     DI,80-4
MOVSW
MOVSW
ADD     SI,NEXTDT-8
MOV     DI,BP
MOV     AX,GREEN
MOV     ES,AX
MOVSW
MOVSW
ADD     DI,80-4
MOVSW
MOVSW
XOR     AL,AL
OUT     0A6H,AL
MOV     AL,CS:MAPEND
AND     AL,AL
JNE     DISM04
INC     BX
```

表画面に背景を描く

裏画面アクセスとする

裏画面に背景を描く

表画面アクセスとする

マップ・データが終わりでなければ
マップ・データアドレスを更新する

```
DISMØ4: ;DISplay Map Ø4
        ADD     BP,4                            表示アドレスの更新
        DEC     CX
        JE      $+5                             CX回ループする
        JMP     DISML1
        NOP
        MOV     CS:EMYSIN,Ø                     CS:EMYSIN←0
        MOV     AX,CS:MPBADR                    AX←マップ・ベース・アドレス
        SUB     AX,2*4                          AX←AX−2×4
        JNB     DISMØ1                          キャリーが立たなければDISM01へ
        MOV     CS:EMYSIN,1                     CS:EMYSIN←1
        CMP     byte ptr CS:MAPEND,ØFFH
        JNE     DISMØ6                          マップ・エンドでなければDISM06へ
        MOV     CS:MAPEND,Ø                     CS:MAPEND←0
        MOV     BX,Ø                            BX←0
        JMP     DISMØ5                          DISM05へ
DISMØ6: ;DISplay Map Ø6
        MOV     BX,CS:MAPADR                    BX←マップ・アドレス
        ADD     BX,2Ø                           マップ・アドレスの更新
        MOV     AX,MPDSEG
        MOV     DS,AX                           マップ・データが終わり
        CMP     [BX],byte ptr ØFFH              でなければDISM05へ
        JNE     DISMØ5
        MOV     BX,Ø                            マップ・アドレスのリセット
DISMØ5: ;DISplay Map Ø5
        MOV     CS:MAPADR,BX                    マップ・アドレスの保存
        MOV     AX,8ØH-2*4                      AX←80H−2×4
DISMØ1: ;DISplay Map Ø1
        MOV     CS:MPBADR,AX                    CS:MPBADR←AX
        POP     DS                              レジスタ復元
        RET                                     リターン

EMYSIN  db      Ø                               EneMY SIgN
EMYNO   db      Ø                               EneMY 番号
ENEMYX  db      Ø                               ENEMY X
MPBADR  dw      8ØH-2*4                         MaP pattern BAse ADdRess
MAPEND  db      Ø                               MAP END
MAPADR  dw      Ø                               MAP ADdRess
KPMAPN  db      Ø                               KeeP MAP pattern 番号
SCLAD1  dw      4ØØØH+SCLPTC                    SCrolL ADdress 1
SCLNO1  dw      Ø                               SCrolL NO 1
SCLNO2  dw      Ø                               SCrolL NO 2

DATLD:  ;DATa LoaD
        MOV     SI,offset LODTBL                SI←ロード情報テーブル先頭アドレス
DLDLP1: ;Data LoaD LooP 1
        MOV     AL,[SI].CMD                     AL←ロード・コマンド
        AND     AL,AL                           コマンド終了か?
        JE      DLDLP2                          コマンド終了であればDLDLP2へ
        CALL    LOAD                            ロード・ルーチン・コール
        ADD     SI,type lodinf                  アドレス更新
        JMP     DLDLP1                          DLDLP1へ
```

```
DLDLP2: ;Data LoaD LooP 2
        MOV     CX,FILEL
        PUSH    DS
        PUSH    ES
        MOV     AX,MPDSEG
        MOV     DS,AX
        MOV     ES,AX
        SHR     CX,1                    敵パターン番号のセット
        MOV     SI,CX
        MOV     DI,8ØØØH
        REP     MOVSB
        POP     ES
        POP     DS
        RET
;
LOAD:   ;LOAD
        LEA     DX,[SI].PASS            ファイルパス名格納アドレス取得
        MOV     AL,Ø                    ファイル・アクセス・コントロール
        MOV     AH,3DH                  ハンドルのオープン
        INT     21H                     ファンクションコール
        JNB     DOPEN                   エラーがなければDOPENへ
        CMP     AX,2                    ファイルが存在しない場合のチェック
        JNE     DLERR                   エラー・コードによる処理の選択
        LEA     DX,[SI].PASS            DX←パスアドレス
        MOV     AH,9                    コマンド・セット
        MOV     [SI].ENDSIN,"$"         ストリング・エンド・コード・セット
        INT     21H                     ファンクションコール
        print   EMES2                   エラー・メッセージ出力
        MOV     AX,ØFFFFH               エラー・サイン・セット
        JMP     DLRET                   リターン
DLERR:  ;Data Load ERRor
        LEA     DX,[SI].PASS            DX←パスアドレス
        MOV     AH,9                    コマンド・セット
        MOV     [SI].ENDSIN,"$"         ストリング・エンド・コード・セット
        INT     21H                     ファンクションコール
        print   EMES1                   エラー・メッセージ出力
        MOV     AX,ØFFFFH               エラー・サイン・セット
        JMP     DLRET                   リターン
DOPEN:  ;Data file OPEN
        MOV     HANDL,AX                ハンドル保存
        MOV     BX,AX                   ハンドルを指定レジスタへ
        MOV     CX,Ø                    CX←0
        MOV     DX,CX                   DX←0
        MOV     AH,42H                  ファイル・ポインタの移動とする
        MOV     AL,2                    ポインタをファイルの終わりに移動とする
        INT     21H                     ファイルの大きさをAXレジスタへ
        MOV     FLLEN,AX                ファイルの大きさを保存
        MOV     BX,HANDL                ハンドルを指定レジスタへ
        MOV     AH,3EH                  ハンドルのクローズ
        INT     21H                     ファンクションコール
        LEA     DX,[SI].PASS            指定ファイルのパス名の格納アドレス取得
```

```
        MOV     AL,Ø            ファイル・アクセス・コントロール
        MOV     AH,3DH          ハンドルのオープン
        INT     21H             ファンクションコール
        MOV     HANDL,AX        ハンドルの保存
        MOV     BX,AX           指定レジスタにハンドルセット
        PUSH    DS              データ・セグメント値をスタックへ退避
        MOV     DX,[SI].LDADR   ロード・アドレス・セット
        MOV     CX,FLLEN        ファイルの大きさをセット
        MOV     AX,[SI].LDSEG   ロード・セグメント値セット
        MOV     DS,AX           DSへAXを介してセグメント値セット
        MOV     AH,3FH          データ・ロード
        INT     21H             ファンクションコール
        POP     DS              DSをスタックから復元
        MOV     FILEL,AX        読み込んだバイト数保存
        MOV     BX,HANDL        ハンドルを指定レジスタへ
        MOV     AH,3EH          ハンドルのクローズ
        INT     21H             ファンクションコール
        XOR     AX,AX           ノーマル・リターンとする
DLRET:  ;Data Load RET
        RET                     リターン

wvram   macro   add,vseg        初期パターン・セット・マクロ定義
        local   IDSM
        MOV     DI,BP           DI←表示アドレス
        MOV     BX,[SI+add]     BX←[SI+add]
        MOV     DX,[SI+add+2]   BX←[SI+add+2]
        MOV     AX,vseg         ┐
        MOV     ES,AX           ┘VRAMセグメントセット
        MOV     CX,2Ø           CX←ループ回数
IDSM:   MOV     ES:[DI],BX      ┐
        MOV     ES:[DI+2],DX    │
        MOV     AL,1            │
        OUT     ØA6H,AL         │表画面と裏画面へ初期の
        MOV     ES:[DI],BX      │背景データを書き込む
        MOV     ES:[DI+2],DX    │
        MOV     AL,Ø            │
        OUT     ØA6H,AL         ┘
        INC     DI              アドレス更新
        INC     DI              アドレス更新
        INC     DI              アドレス更新
        INC     DI              アドレス更新
        LOOP    IDSM            CX回ループ
        endm
;
IDISP:  ;Initial DISPlay
        PUSH    DS              レジスタ保存
        MOV     BP,LVTOP        BP←背景トップアドレス
        MOV     AX,MPPSEG       ┐
        MOV     DS,AX           ┘マップデータ・セグメント・セット
        MOV     CX,4ØØ/32       CX←ループ回数セット
```

```
IDSØØ:  ;Initial DiSplay ØØ
        PUSH    CX                  レジスタ保存
        MOV     CX,32               Y方向ループ数セット
        MOV     SI,5*18ØH           SI←背景パターン番号5のアドレス
IDSØ1:  ;Initial DiSplay Ø1
        PUSH    CX
        wvram   Ø,BLUE
        wvram   NEXTDT,RED
        wvram   NEXTDT*2,GREEN
        ADD     BP,8Ø
        ADD     SI,4
        POP     CX
        DEC     CX
        JE      $+5                 初期画面表示
        JMP     IDSØ1
        POP     CX
        DEC     CX
        JE      $+5
        JMP     IDSØØ
        POP     DS
        RET

FLLEN   dw      Ø                   ファイル名
HANDL   dw      Ø                   ハンドル保存用
FILEL   dw      Ø                   ファイル長さ保存用
EMES1   db      "ファイル オープン エラー$",1Ø,13,"$"
EMES2   db      "ファイル ガ, アリマセン ",1Ø,13,"$"
CLEAR   db      1BH,"[2J",1BH,"[>1h",1BH,"[>5h$"
CSRON   db      1BH,"[>5l",1BH,"[>1l$"
;
CODE    ends

STACK   segment stack
        db      1ØØH    dup (?)
STACK   ends

MPDSEG  segment
        db      ØFFFFH dup (?)
MPDSEG  ends

MPPSEG  segment
        db      18ØH*1ØØ dup (?)
MPPSEG  ends

PTNSEG  segment
        db      2ØØH*1ØØ dup (?)
PTNSEG  ends

MOJSEG  segment
        db      8ØH*5Ø  dup (?)
MOJSEG  ends
```

```
BLUE    segment at ØA8ØØH
        db      ØFFFFH dup (?)
BLUE    ends

RED     segment at ØBØØØH
        db      8ØØØH   dup (?)
RED     ends

GREEN   segment at ØB8ØØH
        db      8ØØØH   dup (?)
GREEN   ends

        end
```

テスト 6-2 テスト・プログラム(TEST 6-2.ASM)

```
;
;****** TEST 6-2 ******
;
CODE    segment                          命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,SS:STACK

print   macro   string                   文字列出力マクロの定義
        LEA     DX,string                マクロパラメータのオフセットを DX へ
        MOV     AH,9                     ファンクションコール番号 9 …… 文字列出力
        INT     21H                      ファンクションコール
        OUT     64H,AL                   GDC にリセット・コマンドを送る
        endm                             マクロ定義終了

PTEST:  ;Program TEST
        CLD                              ストリング命令用フラグを＋方向とする
        MOV     AX,CS                    AX へコードセグメント値をセット
        MOV     DS,AX                    DS へコードセグメント値をセット
        print   CLEAR                    テキスト画面クリア
        CALL    GINIT                    グラフィック・システム初期化
        CALL    DATLD                    データ・ロード
        CALL    CLS                      画面クリア
        print   DSCOR                    得点表示
        MOV     AX,CS                    コード・セグメント値を AX レジスタへ
        MOV     ES,AX                    ES へコード・セグメント値をセット
        MOV     BX,26ØØH*4+226H        } 自機の初期出現位置設定
        MOV     MYLOC,BX               }
```

```
;
        MOV     BX,offset MYBWOK
        MOV     CX,MYBVAL
IMYBLP:;Initialize MY Bullet loop
        MOV     [BX].MB_STATUS,Ø
        ADD     BX,type mb_info
        LOOP    IMYBLP
        CALL    SETINT
        CALL    IDISP
;
MAIN:   ;MAIN loop
        CALL    SSKCK
        CALL    MYBMOV
        CALL    SCROLL
        CALL    MYBMOV
        CALL    MYMOVE
        CALL    PWAIT
        JMP     MAIN
;
EMTTBL  db      54 dup (?)
EMSTBL  db      54 dup (?)
;
SKYP1   equ     4
EXPP1   equ     32
LPS1    equ     4Ø
LPL1    equ     44
;
EMVAL   equ     3Ø
;
eminfo  struc   ;EneMy INFOrmation
STATUS  db      Ø
PATTERN db      Ø
XZAHYOU db      Ø
YZAHYOU db      Ø
POINTER dw      Ø
SCORE   dw      Ø
HOUKOU  db      Ø
EDUMMY  db      7 dup (Ø)
eminfo  ends
;
EMWORK  db      EMVAL * type eminfo dup (Ø)
;
bl_info  struc  ;BuLlet INFOrmation
BL_STATUS   db     Ø
BL_XZAHYOU db      Ø
BL_YZAHYOU db      Ø
BL_HOUKOU  db      Ø
bl_info  ends
;
EMBVAL  equ     4Ø
EMBWOK  db      EMBVAL * type bl_info dup (Ø)
```

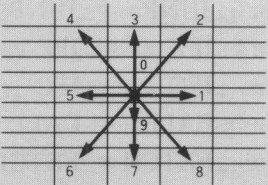
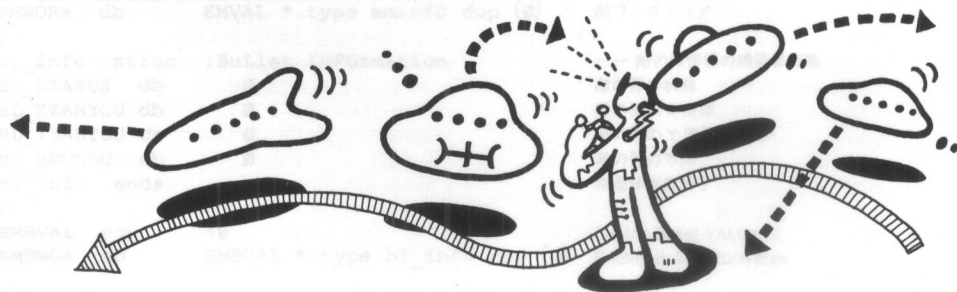
コメント（右欄）:

- 自機の弾のワークエリア初期化（MOV BX〜LOOP IMYBLP）
- 割り込みモード設定（CALL SETINT）
- 初期画面の作成（CALL IDISP）
- SPACE のチェック（CALL SSKCK）
- 自機の弾移動（CALL MYBMOV）
- スクロール（CALL SCROLL）
- 自機の弾移動（CALL MYBMOV）
- 自機の移動（CALL MYMOVE）
- ウェイト（CALL PWAIT）
- MAIN へ（JMP MAIN）
- EneMy Type TaBLe（EMTTBL）
- EneMy Score TaBLe（EMSTBL）
- SKY Pattern 1（SKYP1）
- EXPlosion Pattern 1（EXPP1）
- Land Pattern Small 1（LPS1）
- Land Pattern Large 1（LPL1）
- EneMy Work LENgth（EMVAL）
- —— 敵の情報の構造体定義（eminfo struc）
- 敵の状態（STATUS）
- 敵のパターン番号（PATTERN）
- X 座標（XZAHYOU）
- Y 座標（YZAHYOU）
- ポインタ（POINTER）
- スコア（SCORE）
- 方向（HOUKOU）
- 敵情報のダミー領域（EDUMMY）
- 構造体の定義終了（eminfo ends）
- 敵ワークエリア（EMWORK）
- —— 敵の弾情報の構造体定義（bl_info struc）
- 敵の弾の状態（BL_STATUS）
- 敵の弾の X 座標（BL_XZAHYOU）
- 敵の弾の Y 座標（BL_YZAHYOU）
- 敵の弾の方向（BL_HOUKOU）
- 構造体定義終了（bl_info ends）
- EneMy Bullet VALue（EMBVAL）
- EneMy Bullet Work area（EMBWOK）

```
;
mb_info  struc  ;My Bullet INFOrmation              ── 敵の弾情報の構造体定義
MB_STATUS db     Ø                                  自機の弾の状態
MB_XZAHYOU db    Ø                                  自機の弾のX座標
MB_YZAHYOU db    Ø                                  自機の弾のY座標
mb_info  ends                                       構造体定義終了
;
MYBVAL   equ     12                                 MY Bullet VALue
MYBWOK   db      MYBVAL * type mb_info dup (Ø)      MY Bullet Work area
;
REND     equ     8Ø-4                               Right END
DEND     equ     176                                Down END
;
MYLOC    dw      Ø                                  MY LOCation
MYRST    db      Ø                                  MY ReST
SSKEY    db      ØFFH                               Set Space KEY
;
DSCOR    db      1BH,"[1;1H",1BH,"[21;33mSKY  BRUISER"
MSCOR    db      1BH,"[1;5ØH",1BH,"[23;37m    得点  Ø        "
         db      1BH,"[1;72H",1BH,"[2Ø;32m残り "
MYREM    db      1BH,"[1;77H"
MYREN    db      "8 機",1BH,"[23;37m$"
;
lodinf   struc                                      ロード情報テーブル構造体定義
CMD      db      Ø
PASS     db      "            "
ENDSIN   db      Ø
LDADR    dw      Ø
LDSEG    dw      Ø
lodinf   ends
;
LODTBL: ;LOaD TaBLe
        lodinf <1,"SKYBRU.DAT",Ø,Ø,PTNSEG>
        lodinf <1,"MAPPAT.DAT" ,Ø,Ø,MPPSEG>
        lodinf <1,"MAPDAT2.DAT",Ø,Ø,MPDSEG>
        lodinf < >

        include LIST6-2.ASM
```

3. QRL…パターン・コントロール言語

スクロール・ゲームの醍醐味といえば，やはり次々と襲ってくる謎の飛行物体との，ハデな空中戦ということになりますが，ここでも重要なのは，いかにして敵に生命を与えるかということです．これまでの2つのゲームにおいては，データによる移動および迷路内での追跡という形で，それぞれいちおう敵らしくさせてきました．今回は，これらをさらに一歩進めた形のQRL（Quasi Robot Language＝疑似ロボット言語）と名付けたコマンドで，敵を動かすことにしました．

ロボットといえば，鉄人28号と鉄腕アトムが日本代表（?）として欠かせませんが，この2つはまったく違ったタイプのロボットであるといえます．つまり，鉄人28号のほうはリモコン操作により動かす，いわば操縦者の化身なのですが，アトムのほうは人工知能を持ったロボットなので，持ち主の意志とは違う行動を取ることもあるわけです．どちらも，それぞれに魅力がありますが，これをゲームに当てはめると，キー操作によって動かす主人公は鉄人タイプ，勝手に動く敵はアトム・タイプということができそうです．

いくらアトム・タイプといっても，ここでは弾を発射することと，移動方向を決めることくらいがほとんどで，それ以上の思考力は今後の博士（あなたのことですョ）のアイデアに期待がかかっています．この程度なら，リスト3-4で使われた移動方向データと大差ない，と思った方もいるかもしれません．しかし，このQRLは単なる移動方向データの集まりではなく，実行能力もあるコマンド群なのです．実は，リスト3-5にもNP（New Pointer）という，移動方向データではないコマンドがすでに存在していたことを覚えているでしょうか．QRLとは，この移動方向以外のことをも示す，敵に対するトータル・コマンドと思ってください．では，QRLでどのようなことをさせられるのかは，表6-3に示します．

QRLは大きく分けると，一般コマンドと特殊コマンドから成り立っていますが，一般コマンドでは単に移動方向を示すだけでなく，弾をレベル別（頻度別）に発射させることができます．また，敵の移動方向については，背景スクロールと連動した《方向＝9》が加わっています．移動方向は細かく指定すればするほど，多彩な動きが可能となりますので，興味のある方は確認してみてください．弾の発射に関しては，QRLでは空中敵用（ESHOT1-3）と，地上敵用（LSHOT1-3）の2つを用意しています．こ

	コマンド名	移動方向	弾の発射-1	弾の発射-2	弾の発射-3
			ESHOT 1	ESHOT 2	ESHOT 3
一般コマンド	STOP	00H	＋10H	＋20H	＋30H
	QRR	01H	＋10H	＋20H	＋30H
	QUR	02H	＋10H	＋20H	＋30H
	QUU	03H	＋10H	＋20H	＋30H
	QUL	04H	＋10H	＋20H	＋30H
	QLL	05H	＋10H	＋20H	＋30H
	QDL	06H	＋10H	＋20H	＋30H
	QDD	07H	＋10H	＋20H	＋30H
	QDR	08H	＋10H	＋20H	＋30H
	CHIJOU	09H	＋10H	＋20H	＋30H

4　3　2
0
5　　1
9
6　7　8

移動方向

レベル	確　率
1	5/256
2	32/256
3	256/256

弾の発射レベル

	コマンド形式	コマンド	内　容
特殊コマンド	_END	80H	自爆せよ
	_JUMP〈NP〉	81H	〈NP〉へジャンプ
	_IFZ〈SC〉〈NP〉	82H	〈SC〉をコールしてゼロフラグが立っていれば〈NP〉へジャンプ，立っていなければスキップして次のコマンドへ行け
	_IFC〈SC〉〈NP〉	83H	〈SC〉をコールしてキャリーフラグが立っていれば〈NP〉へジャンプ，立っていなければスキップして次のコマンドへ行け
	_CALL〈SC〉	84H	〈SC〉をコールする
	_FETCH〈SC〉	85H	〈SC〉をコールし得た値（Aレジスタ）を実行する

表6-3　QRLコマンド一覧表

れは発射確率と発射方向(LSHOTは自機を狙う)の違いなのですが，特殊コマンドによって空中敵でもLSHOTを利用することができます．ただ，空中敵の場合は移動方向に合わせたほうが自然なので，本書で作成した空中敵(リスト6-4)はESHOTだけで弾を発射しています．

次に特殊コマンドを見てみましょう．コマンド番号を，わかりやすいようにプログラムで使われているコマンド名(ラベル)で表現すると，次のようになります．

```
80H：_END   =_END    command of QRL
81H：_JUNP  =_JUNP   command of QRL
82H：_IFZ   =_IFZ    command of QRL
83H：_IFC   =_IFC    command of QRL
84H：_CALL  =_CALL   command of QRL
85H：_FETCH =_FETCH  command of QRL
```

このコマンド自体の内容は，表6-3に書かれているとおりで，それほど複雑なことはさせていません．しかし，たったこれだけで何でもできるだけの可能性を秘めているのです．その秘密は，コマンドの次の2バイトをサブ・コマンドとしてコールできる，という所に隠されています．つまり，最終的には移動するための一般コマンドがかならず必要なのですが，その前に一仕事，いや，二仕事でも三仕事でも，好みのことをさせることができるのです．そして，その内容はアイデア次第で，いくらでも拡張することが可能なのが，このQRLの大きな特徴なのです．このリスト6-3においては，基本的なものとして，表6-4のようなサブ・コマンドを用意しましたが，実際に敵の性格に応じていくつか拡張サブ・コマンドを作って，新しい動きをさせています．

サブ・コマンド	内　容（得た値はALregにはいる）
DIRME (DIRection to ME)	敵から見て，自機のいる方向番号を得る
SWINGD (SWING Direction)	敵の現在方向を自機の方へ傾け(0 or ±1)，その方向番号を得る
SAMDIR (SAMe DIRection)	敵の現在方向番号を得る
WHLR (WHich Left or Right)	自機が敵より右にいれば，キャリーフラグを立てて戻る
WHDU (WHich Down or Up)	自機が敵より下にいれば，キャリーフラグを立てて戻る
SHOTAD (SHOT All Direction)	8方向へ弾を撃つ
LSHOT 1 (Land SHOT 1)	乱数値 < 10H で自機に向けて弾を撃つ(更に発射方向別の制限がある)
LSHOT 2 (Land SHOT 2)	乱数値 < 20H で自機に向けて弾を撃つ(更に発射方向別の制限がある)
LSHOT 3 (Land SHOT 3)	乱数値 < 40H で自機に向けて弾を撃つ(更に発射方向別の制限がある)

表6-4 サブ・コマンド一覧表

表6-4には一般コマンドのESHOT1~3は入れてありませんが，当然，これをサブ・コマンドとしても使用できます．また，このサブ・コマンドの内容を見ると，特殊コマンドとの組み合せが大体想像できると思います．さらに，サブ・コマンドから得たデータによって次に何をさせるのか，そこまで想像できるようになれば，もはやQRLは完璧にマスターしたといえるのですが，ここでそこまで理解できなくても問題はありません．次のリスト6-4は，QRLのオン・パレードになっていますから，その時点で自分のモノにすればいいのです．ただし，サブ・コマンドのQRLプログラムについては，短いものばかりですからキチンと内容を確認しておいてください．

今回のプログラムは，ほとんどが敵の動きに終始していましたが，6章のスクロール・ゲームにおいては，スクロール・ルーチンとこのQRLが，マスターしたい2大テーマなのです．ここでのテストでは，QRLとしては見本にもならないような短いものですが，文法的にはこのような形でプログラム(QRLはアセンブラ上で展開する一種の言語ですから，もはやデータとは言わないのです)を作成していくので，自分の手でもう少し変更してから実験してみてください．なお，テストの実行にはリスト6-2をインクルードしていきます．そして，このプログラムも次回(リスト6-4)のために必要ですので，セーブしてください．

テストに際し，マップ・データや敵のデータがあると理想的なのですが，とりあえずは仮のデータでテスト飛行です．

動きを確認したら，いつまでも遊んでないで，最後の大仕事にかからなければなりません．

リスト 6-3 擬似ロボット言語

```
;
;******* List 6-3 *******
;
EMAPP:  ;EneMy APPear
        PUSH    AX                          ┐
        PUSH    BX                          │
        PUSH    CX                          │ レジスタ退避
        PUSH    DI                          │
        PUSH    DS                          ┘
        MOV     AX,CS                       AX←CS
        MOV     DS,AX                       DS←AX
        MOV     BX,offset EMWORK            敵ワークエリア先頭アドレス
        MOV     CX,EMVAL                    敵の総数
EALP:   ;Enemy Appear LooP
        MOV     AL,[BX].STATUS              ┐
        OR      AL,AL                       │ 敵のワークエリアに空きがなければ NEAOK へ
        JNE     NEAOK                       ┘
        MOV     [BX].STATUS,ØFFH            敵出現中のフラグセット
        MOV     AL,EMYNO                    AL←敵パターン番号
        MOV     [BX].PATTERN,AL             敵パターン番号セット
        ADD     AL,AL                       ┐
        MOV     AH,Ø                        │
        MOV     DI,offset EMTTBL            │ 敵ポインタ・セット
        ADD     DI,AX                       │
        MOV     DX,[DI]                     │
        MOV     [BX].POINTER,DX             ┘
        MOV     DI,offset EMSTBL+2          ┐
        ADD     DI,AX                       │ 敵の得点のセット
        MOV     AX,[DI]                     │
        MOV     [BX].SCORE,AX               ┘
        MOV     AL,ENEMYX                   ┐
        MOV     [BX].XZAHYOU,AL             │ 敵の座標セット
        MOV     [BX].YZAHYOU,9              ┘
        MOV     [BX].HOUKOU,Ø
        JMP     EALEX                       EALEX へ
NEAOK:
        ADD     BX,type eminfo              アドレス更新
        LOOP    EALP                        CX 回ループ
EALEX:  ;Enemy Appear Loop EXit
        POP     DS                          ┐
        POP     DI                          │
        POP     CX                          │ レジスタ復元
        POP     BX                          │
        POP     AX                          ┘
        RET                                 リターン
```

```
;
EMMVAL: ;EneMy   MoVe ALl
        MOV      SI,offset EMWORK         ┐
        MOV      CX,EMVAL                 │
EMMLP:  ;EneMy Move LooP                  │
        MOV      AL,[SI].STATUS           │
        OR       AL,AL                    │
        JE       EMNEXT                   │
        PUSH     CX                       │
        PUSH     SI                       │ すべての敵をコマンドに従い移動する
        CALL     EMMOVE                   │
        POP      SI                       │
        POP      CX                       │
EMNEXT: ;EneMy NEXT                       │
        ADD      SI,type eminfo           │
        LOOP     EMMLP                    │
        RET                               ┘
;
EMMOVE: ;EneMy MOVE
        INC      AL                       ┐
        JNE      NENEMY                   │ AL=FFHならENEMYへ
        JMP      ENEMY                    ┘
NENEMY: ;Not ENEMY
        MOV      CL,[SI].XZAHYOU            敵のX座標
        MOV      CH,[SI].YZAHYOU            敵のY座標
        INC      [SI].PATTERN               爆発パターンを+1する
        INC      AL                       ┐ AL=FEHならLANDDへ
        JNE      LANDD                    ┘
        MOV      AL,[SI].PATTERN            AL←パターン番号
        CMP      AL,LDEADP                ┐ AL>LAEADPならNEDISPへ
        JNB      NEDISP                   ┘
        INC      CH                         CH←CH+1
        MOV      [SI].YZAHYOU,CH            Y座標保存
        JMP      DISP                       DISPへ
NEDISP: ;Not Enemy DISP
        JMP      EMFIN                      EMFINへ
LANDD:  ;LAND Dead
        CMP      CH,DEND                  ┐
        JB       NEMFIND                  │ AL=DENDならEMFINへ
        JMP      EMFIN                    ┘
NEMFIND: ;Not EMFIN
        INC      CH                         CH←CH+1
        MOV      [SI].YZAHYOU,CH            Y座標保存
        MOV      AL,[SI].PATTERN            AL←パターン番号
        CMP      AL,LDEADP                  AL=LDEADPか?
        JE       LDPNOK                     AL=LDEADPならLDPNOKへ
        JMP      DISP                       DISPへ
LDPNOK: ;Land Dead Pattern OK
        MOV      [SI].STATUS,Ø              出現フラグ・リセット
        MOV      AL,39                    ┐ 爆発跡パターン番号セット
        MOV      [SI].PATTERN,AL          ┘
```

```
LDISP:  ;Land DISP
        PUSH    SI              レジスタ保存
        PUSH    ES
        CALL    XYADR           表示アドレスを求める
        CALL    PDADR           パターン格納アドレスを求める
        MOV     SI,BP           SI←パターン格納アドレス
        MOV     DI,DISPAD       DI←表示アドレス
        PUSH    DS              レジスタ退避
        MOV     AX,GREEN
        MOV     ES,AX
        MOV     CH,20H
LDLP1:  ;Land Dead LooP 1
        MOV     CL,2
LDLP2:  ;Land Dead LooP 2
        MOV     AX,BLUE
        MOV     DS,AX
        MOV     AX,1
        OUT     0A6H,AL
        MOV     DX,[DI]
        MOV     BX,[DI+8000H]
        MOV     BP,ES:[DI]
        MOV     AX,PTNSEG
        MOV     DS,AX
        MOV     AX,[SI]
        AND     DX,AX
        AND     BX,AX
        AND     BP,AX
        OR      DX,[SI+NEXTDT]
        OR      BX,[SI+2*NEXTDT]
        OR      BP,[SI+3*NEXTDT]        地上の敵の爆発跡の表示
        MOV     AX,BLUE
        MOV     DS,AX
        MOV     [DI],DX
        MOV     [DI+8000H],BX
        MOV     ES:[DI],BP
        XCHG    AL,AH
        OUT     0A6H,AL
        MOV     [DI],DX
        MOV     [DI+8000H],BX
        MOV     ES:[DI],BP
        ADD     DI,2
        ADD     SI,2
        DEC     CL
        JNE     LDLP2
        ADD     DI,HLEN-4
        CMP     DI,VEND
        JL      LD003
        SUB     DI,VEND
LD003:  ;Land dead 003
        DEC     CH
        JNE     LDLP1
        POP     DS
```

```
        POP     ES                      } レジスタ復元
        POP     SI
        RET                             リターン
;
ENEMY:  ;ENEMY
        MOV     BX,[SI].POINTER         BX←コマンド・ポインタ
COM:    ;COMmand
        MOV     CL,[BX]                 CL←コマンド
        MOV     AL,CL                   }
        AND     AL,8ØH                  } コマンドのビット7が0ならGENCOMへ
        JNE     NGENCOM                 }
        JMP     GENCOM                  }
NGENCOM:;Not GENCOM
        MOV     AL,CL                   }
        AND     AL,7FH                  } ALが0でなければSPCOMへ
        JNE     SPCOM                   }
        MOV     [SI].STATUS,ØFEH        自爆コマンド・セット
        MOV     AL,EXPP1                AL←爆発パターン番号
        MOV     [SI].PATTERN,AL         パターン番号セット
        MOV     CL,[SI].XZAHYOU         CL←X座標
        MOV     CH,[SI].YZAHYOU         CH←Y座標
        INC     CH                      CH←CH+1
        MOV     [SI].YZAHYOU,CH
        JMP     DISP                    DISPへ
;
SPCOM:  ;SPecial COMmand                ── 特殊コマンド
        DEC     AL                      }
        JNE     NGETNP                  } AL=1なら_JUMP
        JMP     GETNP                   }
NGETNP: ;Not GETNP
        INC     BX                      }
        MOV     DX,[BX]                 } DX←コマンドの次の2バイト・データ
        INC     BX                      }
        PUSH    BX                      コマンド・ポインタを退避
        XCHG    DX,BX                   DX←→BX
        DEC     AL                      }
        JNE     NCZGNP                  } AL=2(_IFZ)ならCZGNPへ
        JMP     CZGNP                   }
NCZGNP: ;Not CZGNP
        DEC     AL                      }
        JNE     NCCGNP                  } AL=3(_IFC)ならCCGNPへ
        JMP     CCGNP                   }
NCCGNP: ;Not CCGNP
        DEC     AL                      }
        JNE     CGGCO                   } AL=4(_CALL)ならCONLYへ
        JMP     CONLY                   }
CGGCO:  ;Command GGCO
        MOV     CX,offset RET1          } BXで示されるアドレスへジャンプし
        PUSH    CX                      } RETでRET1へ戻る
        JMP     BX                      }
```

```
RET1:   ;RETurn 1
        POP     BX              コマンド・ポインタを取り出す
        MOV     CL,AL           CL ← AL …… コール先で得た新コマンド
        JMP     GENCOM
;
CONLY:  ;CONLY
        MOV     CX,offset RET2
        PUSH    CX              CALL BX
        JMP     BX
RET2:   ;RETurn 2
        POP     BX
        INC     BX              コマンド・ポインタを取り出す
        JMP     COM
;
CCGNP:
        MOV     CX,offset RET3
        PUSH    CX              CALL BX
        JMP     BX
RET3:   ;RETurn 3
        POP     BX              コマンド・ポインタを取り出す
        JNB     NGETNPØ         キャリーが立っていれば GETNP へ
        JMP     GETNP
NGETNPØ:;Not GETNPØ
        ADD     BX,3            コマンド・ポインタ更新
        JMP     COM             COM へ
;
CZGNP:
        MOV     CX,offset RET4
        PUSH    CX              CALL BX
        JMP     BX
RET4:   ;RETurn 4
        POP     BX              コマンド・ポインタを取り出す
        JNE     NGETNP2         ゼロ・フラグが立っていれば GETNP へ
        JMP     GETNP
NGETNP2:;Not GETNP2
        ADD     BX,3            コマンド・ポインタ更新
        JMP     COM             COM へ
;
GETNP:  ;GET New Pointer
        INC     BX
        MOV     DX,[BX]         新コマンド・ポインタを得る
        INC     BX
        XCHG    DX,BX
        JMP     COM             COM へ
;
GENCOM: ;GENeral COMmand        一般コマンド
        INC     BX
        MOV     [SI].POINTER,BX コマンド・ポインタ更新
        MOV     AL,CL           AL ← CL
        AND     AL,3ØH          ビット 4,5 以外のビットは 0 にする
```

```
        JNE     NONLYM                   } 0ならONLYMへ
        JMP     ONLYM
NONLYM: ;Not ONLYM
        MOV     BX,offset ONLYM          } 下記ジャンプ先でRETがあれば,
        PUSH    BX                         すべてONLYMへ戻るようにする
        CMP     AL,10H
        JNE     NESHOT1                  } AL=10HならESHOT1へ
        JMP     ESHOT1
NESHOT1:;Not ESHOT1
        CMP     AL,20H
        JNE     NESHOT2                  } AL=10HならESHOT2へ
        JMP     ESHOT2
NESHOT2:;Not ESHOT2
        JMP     ESHOT3                     ESHOT3へ
;
ONLYM:  ;ONLY Move
        MOV     AL,CL                      AL←コマンド
        MOV     CL,[SI].XZAHYOU            CL←X座標
        MOV     CH,[SI].YZAHYOU            CL←Y座標
        AND     AL,0FH                     移動方向を取り出す
        MOV     [SI].HOUKOU,AL             移動方向の保存
        MOV     AH,0
        SHL     AX,1
        MOV     BX,offset EMTBL          } 移動方向別のジャンプ
        ADD     BX,AX
        JMP     [BX]
;
EMTBL   dw      offset EMSTOP              移動方向別ジャンプ・テーブル
        dw      offset EMRR
        dw      offset EMUR
        dw      offset EMUU
        dw      offset EMUL
        dw      offset EMLL
        dw      offset EMDL
        dw      offset EMDD
        dw      offset EMDR
        dw      offset ECHIJ
;
EMSTOP: ;EneMy STOP
        CMP     CH,DEND
        JB      $+5                      } CH=DENDならEMFINへ
        JMP     EMFIN
        INC     CH                         CH←CH+1
        PUSH    CX
        MOV     BX,402H
        CALL    CLPTXY                   } 移動方向=0の場合の不要部分消去+次座標計算
        POP     CX
        INC     CH
        JMP     EMDISP
```

```
;
EMRR:   ;EneMy direction = QRR
        MOV     AL,REND
        CMP     AL,CL                 CL=REND なら EMFIN へ
        JNE     NEMFIN
        JMP     EMFIN
NEMFIN: ;Not EMFIN
        CMP     CH,DEND
        JB      NEMFINØ               CH=DEND なら EMFIN へ
        JMP     EMFIN
NEMFINØ:;Not EMFINØ
        INC     CH
        PUSH    CX
        MOV     BX,12ØH
        CALL    CLPTXY
        POP     CX
        PUSH    CX
        INC     CL                    移動方向=1 の場合の不要部分消去＋次座標計算
        MOV     BX,3Ø2H
        CALL    CLPTXY
        POP     CX
        INC     CH
        INC     CL
        JMP     EMDISP
;
EMUR:   ;EneMy direction = QUR
        MOV     AL,REND
        CMP     AL,CL                 CL=REND なら EMFIN へ
        JNE     NEMFIN1
        JMP     EMFIN
NEMFIN1:;Not EMFIN1
        CMP     CH,1Ø
        JNB     NEMFIN2               CH=10(上エンド)なら EMFIN へ
        JMP     EMFIN
NEMFIN2:;Not EMFIN2
        PUSH    CX
        ADD     CH,14
        MOV     BX,4Ø6H
        CALL    CLPTXY
        POP     CX
        PUSH    CX
        INC     CH                    移動方向=2 の場合の不要部分消去＋次座標計算
        MOV     BX,11AH
        CALL    CLPTXY
        POP     CX
        INC     CL
        SUB     CH,2
        JMP     EMDISP
```

```
;
EMUU:   ;EneMy direction = QUU
        CMP     CH,1Ø
        JNB     NEMFIN3
        JMP     EMFIN
NEMFIN3:;Not EMFIN3
        PUSH    CX
        ADD     CH,14
        MOV     BX,4Ø6H
        CALL    CLPTXY
        POP     CX
        SUB     CH,2
        JMP     EMDISP
;
EMUL:   ;EneMy direction = QUL
        XOR     AL,AL
        CMP     AL,CL
        JNE     NEMFIN4
        JMP     EMFIN
NEMFIN4:;Not EMFIN4
        CMP     CH,1Ø
        JNB     NEMFIN5
        JMP     EMFIN
NEMFIN5:;Not EMFIN5
        PUSH    CX
        INC     CH
        ADD     CL,3
        MOV     BX,11AH
        CALL    CLPTXY
        POP     CX
        PUSH    CX
        ADD     CH,14
        MOV     BX,4Ø6H
        CALL    CLPTXY
        POP     CX
        DEC     CL
        SUB     CH,2
        JMP     EMDISP
;
EMLL:   ;EneMy direction = QLL
        XOR     AL,AL
        CMP     AL,CL
        JNE     NEMFIN6
        JMP     EMFIN
NEMFIN6:;Not EMFIN6
        CMP     CH,DEND
        JB      NEMFIN7
        JMP     EMFIN
```

CH=10(上エンド)ならEMFINへ

移動方向=3の場合の不要部分消去+次座標計算

CL=0(左エンド)ならEMFINへ

CH=10(上エンド)ならEMFINへ

移動方向=4の場合の不要部分消去+次座標計算

CL=0(左エンド)ならEMFINへ

CH=DENDならEMFINへ

```
NEMFIN7:;Not EMFIN7
        INC     CH
        PUSH    CX
        ADD     CL,3
        MOV     BX,120H
        CALL    CLPTXY
        POP     CX
        PUSH    CX
        MOV     BX,302H
        CALL    CLPTXY
        POP     CX
        INC     CH
        DEC     CL
        JMP     EMDISP
;
EMDL:   ;EneMy direction = QDL
        CMP     CH,DEND
        JB      NEMFIN8
        JMP     EMFIN
NEMFIN8:;Not EMFIN8
        XOR     AL,AL
        CMP     AL,CL
        JNE     NEMFIN9
        JMP     EMFIN
NEMFIN9:;Not EMFIN9
        INC     CH
        PUSH    CX
        MOV     BX,406H
        CALL    CLPTXY
        POP     CX
        PUSH    CX
        ADD     CL,3
        ADD     CH,3
        MOV     BX,11AH
        CALL    CLPTXY
        POP     CX
        DEC     CL
        ADD     CH,3
        JMP     EMDISP
;
EMDD:   ;EneMy direction = QDD
        CMP     CH,DEND
        JB      NEMFINA
        JMP     EMFIN
NEMFINA:;Not EMFINA
        INC     CH
        PUSH    CX
        MOV     BX,406H
        CALL    CLPTXY
        POP     CX
        ADD     CH,3
        JMP     EMDISP
```

移動方向＝5の場合の不要部分消去＋次座標計算

CH＝DENDならEMFINへ

CL＝0(左エンド)ならEMFINへ

移動方向＝6の場合の不要部分消去＋次座標計算

CH＝DENDならEMFINへ

移動方向＝7の場合の不要部分消去＋次座標計算

```
;
EMDR:    ;EneMy direction = QDR
         MOV      AL,REND                   CL=REND なら EMFIN へ
         CMP      AL,CL
         JNE      NEMFINB
         JMP      EMFIN
NEMFINB:;Not EMFINB
         CMP      CH,DEND                   CH=DEND なら EMFIN へ
         JB       NEMFINC
         JMP      EMFIN
NEMFINC:;Not EMFINC
         INC      CH                        移動方向=8 の場合の不要部分消去+次座標計算
         PUSH     CX
         MOV      BX,120H
         CALL     CLPTXY
         POP      CX
         INC      CL
         PUSH     CX
         MOV      BX,306H
         CALL     CLPTXY
         POP      CX
         ADD      CH,3
EMDISP: ;EneMy DISPlay
         MOV      [SI].XZAHYOU,CL           主人公の弾との衝突チェック
         MOV      [SI].YZAHYOU,CH
         PUSH     CX
         CALL     MYBCHK
         POP      CX
         MOV      AL,[SI].PATTERN
         JMP      DISP
;
ECHIJ: ;EneMy CHIJou
         CMP      CH,DEND                   CH=DEND なら EMFIN へ
         JNB      EMFIN
         INC      CH                        CH←CH+1
         JMP      EMDISP                    EMDISP へ
;
EMFIN:   ;EneMy FINish
         MOV      [SI].STATUS,0             出現フラグをリセット
         INC      CH                        アドレス更新
         MOV      BX,420H                   消去サイズ
         JMP      CLPTXY                    CLPTXY へ
;
MYBCHK: ;MY Bullet CHecK with enemy
         MOV      CH,MYBVAL                 CH←自機の弾の総数
         MOV      BX,offset MYBWOK          BX←自機の弾のワークエリア先頭アドレス
MBCLP:   ;MY Bullet Check LooP
         MOV      AL,[BX].MB_STATUS         自機の弾が出現していなければ NTMB1 へ
         OR       AL,AL
         JNE      NNTMB2
         JMP      NTMB1
```

```
NNTMB2: ;Not NTMB2
        MOV     AL,[BX].MB_XZAHYOU
        SUB     AL,[SI].XZAHYOU        自機の弾のX座標－敵のX座標≧4ならNTMB1へ
        CMP     AL,4
        JNB     NTMB1
NNTMB1: ;Not NTMB1
        MOV     AL,[BX].MB_YZAHYOU
        SUB     AL,[SI].YZAHYOU        自機の弾のY座標－敵のY座標≧12ならNTMB1へ
        CMP     AL,12
        JNB     NTMB1
        MOV     AL,[SI].PATTERN        敵のパターン番号≧4ならAL←FEH
        CMP     AL,4                   空中敵爆発フラグ
        MOV     AL,ØFEH                敵のパターン番号<4ならAL←FDH
        JNB     EXPSET                 地上敵爆発フラグ
        DEC     AL
EXPSET: ;EXPlosion SET
        PUSH    BX                     レジスタ退避
        PUSH    SI
        MOV     [SI].STATUS,AL         爆発フラグ・セット
        MOV     [SI].PATTERN,EXPP1     爆発パターン・セット
        MOV     DX,[SI].SCORE          得点の加算
        CALL    DISPSC
        MOV     byte ptr BEEP,1        BEEPサイン・オン
        POP     SI                     レジスタ復元
        POP     BX
        MOV     [BX].MB_STATUS,Ø       自機の出現フラグ・リセット
        RET
NTMB1:  ;NexT My Bullet 1
        ADD     BX,type mb_info        次の自機の弾のワークエリア
        DEC     CH
        JNE     MBCLP
        RET
;
EMBMOV: ;EneMy Bullet MOVe             ―― 敵の弾の移動
        MOV     BX,offset EMBWOK       BX←敵の弾のワークエリア先頭アドレス
        MOV     CH,EMBVAL              CH←敵の弾の総数
EBMLP:  ;Enemy Bullet Move LooP
        MOV     AL,[BX].BL_STATUS
        OR      AL,AL                  敵の弾が出現中ならEBINGをコール
        JE      $+5
        CALL    EBING
        ADD     BX,type bl_info
        DEC     CH                     アドレス更新
        JNE     EBMLP
        RET
;
EBING:  ;Enemy Bullet is movING
        PUSH    CX
        MOV     CL,[BX].BL_XZAHYOU     CL←X座標
        MOV     CH,[BX].BL_YZAHYOU     CH←Y座標
```

```
        PUSH    BX
        PUSH    CX
        MOV     BX,1Ø8H + SCLDOT
        CALL    CLPTXY
        POP     CX
        POP     BX
        MOV     AL,[BX].BL_HOUKOU
        MOV     AH,Ø
        SHL     AX,1
        PUSH    BX
        MOV     BX,offset EMBTBL
        ADD     BX,AX
        JMP     [BX]
;
EMBTBL  dw      offset EMBDD
        dw      offset EMBRR
        dw      offset EMBUR
        dw      offset EMBUU
        dw      offset EMBUL
        dw      offset EMBLL
        dw      offset EMBDL
        dw      offset EMBDD
        dw      offset EMBDR
;
EMBRR:  ;EneMy Bullet direction = QRR
        CMP     CL,REND+3
        JNE     NEMBFI1
        JMP     EMBFIN
NEMBFI1:;Not EMBFI1
        CMP     CH,DEND+12
        JB      NEMBFI2
        JMP     EMBFIN
NEMBFI2:;Not EMBFI2
        INC     CL
        INC     CH
        JMP     EBDISP
;
EMBUR:  ;EneMy Bullet direction = QUR
        CMP     CH,13
        JNB     NEMBFI3
        JMP     EMBFIN
NEMBFI3:;Not EMBFI3
        CMP     CL,REND+3
        JNE     NEMBFI4
        JMP     EMBFIN
NEMBFI4:;Not EMBFI4
        INC     CL
        SUB     CH,3
        JMP     EBDISP
```

(CL, CH)にある敵の弾を消去

移動方向に従ってジャンプする

敵の弾の方向別ジャンプ・テーブル

CL＝右エンド+3ならEMBFINへ

CH＝下エンド+6ならEMBFINへ

(CL, CH)→(CL+1, CH+1)

CH＝上エンドならEMBFINへ

CL＝右エンド+3ならEMBFINへ

(CL, CH)→(CL+1, CH−3)

```
;
EMBUU:  ;EneMy Bullet direction = QUU
        CMP     CH,13
        JNB     NEMBFI5                 } CH＝上エンドならEMBFINへ
        JMP     EMBFIN
NEMBFI5:;Not EMBFI5
        SUB     CH,3                    } (CL, CH)→(CL, CH−3)
        JMP     EBDISP
;
EMBUL:  ;EneMy Bullet direction = QUL
        CMP     CH,13
        JNB     NEMBFI6                 } CH＝上エンドならEMBFINへ
        JMP     EMBFIN
NEMBFI6:;Not EMBFI6
        AND     CL,CL
        JNE     NEMBFI7                 } CL＝左エンドならEMBFINへ
        JMP     EMBFIN
NEMBFI7:;Not EMBFI7
        DEC     CL
        SUB     CH,3                    } (CL, CH)→(CL−1, CH−3)
        JMP     EBDISP
;
EMBLL:  ;EneMy Bullet direction = QLL
        AND     CL,CL
        JNE     NEMBFI8                 } CL＝左エンドならEMBFINへ
        JMP     EMBFIN
NEMBFI8:;Not EMBFI8
        CMP     CH,DEND+12
        JB      NEMBFI9                 } CH＝下エンド+12ならEMBFINへ
        JMP     EMBFIN
NEMBFI9:;Not EMBFI9
        DEC     CL
        INC     CH                      } (CL, CH)→(CL−1, CH+1)
        JMP     EBDISP
;
EMBDL:  ;EneMy Bullet direction = QDL
        AND     CL,CL
        JNE     NEMBFIA                 } CL＝左エンドならEMBFINへ
        JMP     EMBFIN
NEMBFIA:;Not EMBFIA
        CMP     CH,DEND+12
        JB      NEMBFIB                 } CH＝下エンド+12ならEMBFINへ
        JMP     EMBFIN
NEMBFIB:;Not EMBFIB
        DEC     CL
        ADD     CH,5                    } (CL, CH)→(CL−1, CH+5)
        JMP     EBDISP
```

```
;
EMBDD:  ;EneMy Bullet direction = QDD
        CMP     CH,DEND+12          CH＝下エンド＋12 なら EMBFIN へ
        JB      NEMBFIC
        JMP     EMBFIN
NEMBFIC:;Not EMBFIC
        ADD     CH,5                (CL, CH)→(CL, CH＋5)
        JMP     EBDISP
;
EMBDR:  ;EneMy Bullet direction = QDR
        CMP     CH,DEND+12          CH＝下エンド＋12 なら EMBFIN へ
        JB      NEMBFID
        JMP     EMBFIN
NEMBFID:;Not EMBFID
        CMP     CL,REND+3           CL＝右エンド＋3 なら EMBFIN へ
        JNE     NEMBFIE
        JMP     EMBFIN
NEMBFIE:;Not EMBFIE
        INC     CL                  (CL, CH)→(CL＋1, CH＋5)
        ADD     CH,5
EBDISP: ;EneMy Bullet FINish
        POP     BX
        MOV     [BX].BL_XZAHYOU,CL  座標保存
        MOV     [BX].BL_YZAHYOU,CH
        PUSH    BX                  (CL, CH)に弾を表示
        CALL    BLPUT
        POP     BX
        POP     CX
        RET
;
EMBFIN: ;EneMy Bullet FINish
        POP     BX                  敵の弾の出現フラグを 0 にする
        MOV     [BX].BL_STATUS,Ø
        POP     CX
        RET
;
ESHOT1: ;Enemy SHOT 1
        CALL    RND                 乱数値＜5 なら ESHOT3 へ
        CMP     AL,5                        弾発射の確率← 5/256
        JB      ESHOT3
        RET
ESHOT2: ;Enemy SHOT 2
        CALL    RND                 乱数値＜20H なら ESHOT3 へ
        CMP     AL,2ØH                      弾発射の確率← 32/256
        JB      $+3
        RET
ESHOT3: ;Enemy SHOT 3
        MOV     CH,EMBVAL           CH ←敵の弾の総数
        MOV     BX,offset EMBWOK    BX ←敵の弾のワークエリアの先頭アドレス
        MOV     DX,type bl_info     DX ←敵の弾 1 つのワークエリアの長さ
```

```
ESLP:    ;Enemy Shot LooP
         MOV      AL,[BX].BL_STATUS
         OR       AL,AL               ワークエリアに空きがあればESOKへ
         JE       ESOK
         ADD      BX,DX               アドレス更新
         DEC      CH
         JNE      ESLP                敵の弾数だけループする
         RET
ESOK:    ;Enemy Shot OK
         MOV      AL,[SI].HOUKOU
         CMP      AL,5                AL<5ならED1234へ
         JB       ED1234
         CMP      AL,7                AL<7ならED56へ
         JB       ED56
         MOV      DX,6Ø2H             弾出現座標のオフセット値
         JMP      DTSET
ED56:    ;Enemy Direction=5,6
         MOV      DX,4ØØH             弾出現座標のオフセット値
         JMP      DTSET
ED1234:  ;Enemy Direction=1,2,3,4
         CMP      AL,3                AL<3ならED12へ
         JB       ED12
         MOV      DX,ØØ1H             弾出現座標のオフセット値
         JMP      DTSET
ED12:    ;Enemy Direction=5,6
         MOV      DX,3Ø3H             弾出現座標のオフセット値
;
DTSET:   ;DaTa SET
         MOV      [BX].BL_STATUS,1
         ADD      DL,[SI].XZAHYOU
         MOV      [BX].BL_XZAHYOU,DL  初期データ・セット
         ADD      DH,[SI].YZAHYOU
         MOV      [BX].BL_YZAHYOU,DH
         MOV      [BX].BL_HOUKOU,AL
         RET
;
DIRME:   ;DIRection to ME             ── 自機のいる方向番号を得る
         MOV      BX,[MYLOC]          BX←自機の座標
         MOV      AL,[SI].XZAHYOU     AL←弾のX座標
         SUB      AL,BL               X軸の差
         JB       ERIGHT              キャリーが立てばERIGHTへ
         JMP      ELEFT               ELEFTへ
ERIGHT:  ;Enemy'S RIGHT
         NEG      AL                  AL←-AL
         MOV      CL,AL               CL←AL
         MOV      AL,[SI].YZAHYOU     AL←Y座標
         SUB      AL,BH               Y軸の差
         JNB      ERNU                キャリーが立たなければERNUへ
ERND:    ;Enemy's Right aNd Down
         NEG      AL
         MOV      CH,AL               Y軸の差の絶対値
```

```
        MOV     AL,CL                  ┐
        ADD     AL,AL                  │ CL×4 < CH なら,AL←7としてリターン
        ADD     AL,AL                  │    ↑        ↑
        CMP     AL,CH                  │ X軸の差    Y軸の差
        MOV     AL,7                   │
        JNB     $+3                    │
        RET                            ┘
        MOV     AL,CL                  ┐
        ADD     AL,AL                  │
        ADD     AL,CL                  │ CL×3 < CH×2 なら,AL←8としてリターン
        SAL     CH,1                   │    ↑         ↑
        CMP     AL,CH                  │ X軸の差    Y軸の差
        MOV     AL,8                   │
        JNB     $+3                    │
        RET                            ┘
        MOV     AL,1                   AL←1としてリターン
        RET
ERNU:   ;Enemy's Right aNd Up
        SAL     CL,1                   ┐
        CMP     AL,CL                  │ AL < C×2 なら AL←2としてリターン
        MOV     AL,2                   │ ↑        ↑
        JNB     $+3                    │ Y軸の差  X軸の差
        RET                            ┘
        INC     AL                     AL←3としてリターン
        RET
;
ELEFT:  ;Enemy's LEFT
        MOV     CL,AL                  X軸の差
        MOV     AL,[SI].YZAHYOU        敵のY座標
        SUB     AL,BH                  Y軸の差
        JB      ELND                   AL≧0 なら ELNU へ
        JMP     ELNU
ELND:   ;Enemy's Left aNd Down         Y軸の差の絶対値
        NEG     AL
        MOV     CH,AL                  ┐
        MOV     AL,CL                  │
        ADD     AL,AL                  │ CL×4 < CH なら,AL←7としてリターン
        ADD     AL,AL                  │    ↑        ↑
        CMP     AL,CH                  │ X軸の差    Y軸の差
        MOV     AL,7                   │
        JNB     $+3                    │
        RET                            ┘
        MOV     AL,CL                  ┐
        ADD     AL,AL                  │
        ADD     AL,CL                  │
        SAL     CH,1                   │ CL×3 < CH×2 なら,AL←6としてリターン
        CMP     AL,CH                  │    ↑         ↑
        MOV     AL,6                   │ X軸の差    Y軸の差
        JNB     $+3                    │
        RET                            ┘
```

```
        DEC     AL                      AL←5としてリターン
        RET
ELNU:   ;Enemy's Left aNd Up
        SAL     CL,1
        CMP     AL,CL                   AL < CL×2 ならAL←4としてリターン
        MOV     AL,4                    ↑         ↑
        JNB     $+3                     X軸の差   Y軸の差
        RET
        DEC     AL                      AL←3としてリターン
        RET
;
SWINGD: ;SWING Direction                ── 追跡する方向を示す値を得る
        CALL    DIRME                   AL←自機のいる方向番号
        MOV     CH,[SI].HOUKOU          敵の移動方向(0もある)
        SUB     AL,CH                   AL←AL−CH
        JE      NOSWG                   AL=0ならNOSWGへ
        JS      SWGM                    AL<0ならSWGMへ
        CMP     AL,4
        JNB     DECSWG                  AL≧4ならDECSWGへ
        JMP     INCSWG
SWGM:   ;SWinG Minus
        CMP     AL,-4
        JNB     DECSWG                  AL≧−4ならDECSWGへ
INCSWG: ;INCrement SWinG                ── AL=1,2,3,−5,−6,−7のとき
        MOV     AL,CH
        AND     AL,7                    AL←CH+1 …… AL=1〜8になるよう処理
        INC     AL
        RET
DECSWG: ;DECrement SWinG                ── AL=4,5,6,7,8,−1,−2,−3,−4のとき
        MOV     AL,CH
        DEC     AL
        AND     AL,7
        JE      $+3                     AL←CH+1 …… AL=1〜8になるよう処理
        RET
        MOV     AL,8
        RET
NOSWG:  ;NO SWing
        MOV     AL,CH                   方向の変化なし
        RET
;
SAMDIR: ;SAMe DIRection                 ── 敵の方向番号を得る
        MOV     AL,[SI].HOUKOU          敵の移動方向
        RET
;
WHLR:   ;WHich Left or Right
        MOV     AL,byte ptr MYLOC
        CMP     AL,[SI].XZAHYOU         自機が敵より右にいれば
        CMC                             キャリーフラグが立つ
        RET
```

```
;
WHDU:    ;WHich Down or Up
         MOV      AL,byte ptr MYLOC+1     自機が敵より下にいれば
         CMP      AL,[SI].YZAHYOU         キャリーフラグが立つ
         RET
;
SHOTAD: ;SHOT All Direction
         MOV      CL,8                    CL ← 8 …… 敵の弾発射の数, および方向を示す
SADLP:
         MOV      BDFIXW,CL               弾の発射予定方向設定
         PUSH     CX                      敵の弾のワークエリアに空きがあれば, 発射の
         CALL     BDFIX                   設定をし, なければキャリーフラグが立つ
         POP      CX
         JNB      $+3                     キャリーが立っていればリターン
         RET
         DEC      CL                      CL 回ループする
         JNE      SADLP
         RET
;
LSHOT1: ;Land SHOT 1
         CALL     RND
         CMP      AL,10H
         JB       $+3                     乱数値≧10H ならリターン
         RET
         JMP      PBYD
;
LSHOT2: ;Land SHOT 2
         CALL     RND
         CMP      AL,20H
         JB       $+3                     乱数値≧20H ならリターン
         RET
         JMP      PBYD
;
LSHOT3: ;Land SHOT 3
         CALL     RND
         CMP      AL,40H                  乱数値≧40H ならリターン
         JB       $+3
         RET
;
PBYD:    ;Possibility BY Direction
         CALL     DIRME                   AL ← 自機のいる方向番号となる
         MOV      BDFIXW,AL               弾の発射予定方向設定
         AND      AL,7
         SUB      AL,3
         JNB      CKPOS                   AL=1,2,3,4,5,6,7,8 が
         NEG      AL                             ↓
CKPOS:   ;ChecK POSsibility               BL=3,2,1,2,3,4,5,4 となる
         INC      AL
         MOV      BL,AL
         CALL     RND                     AL ← 1〜3 の乱数
         AND      AL,3
```

```
        CMP     AL,BL           BL≦AL ならリターン……フィルタ①
        JB      $+3
        RET
        CALL    RND
        RCL     AL,1            乱数値≧80H ならリターン …… フィルタ②
        JNB     $+3
        RET
BDFIX:  ;Bullet Direction FIX
        MOV     AL,BDFIXW       AL ←弾の発射予定方向
        CMP     AL,5            AL<5 なら BD1234 へ
        JB      BD1234
        CMP     AL,7            AL<7 なら BD56 へ
        JB      BD56
        MOV     BX,6Ø2H         弾出現座標のオフセット値
        JMP     BPSET
BD56:   ;Bullet Direction 5,6
        MOV     BX,4ØØH         弾出現座標のオフセット値
        JMP     BPSET
BD1234: ;Bullet Direction 1,2,3,4
        CMP     AL,3            AL<3 なら BD12 へ
        JB      BD12
        MOV     BX,ØØ1H         弾出現座標のオフセット値
        JMP     BPSET
BD12:   ;Bullet Direction 1,2
        MOV     BX,3Ø3H         弾出現座標のオフセット値
BPSET:  ;Bullet Position SET
        MOV     SETPDD,BX
        MOV     CL,AL           弾の発射予定方向
;
SKSWK:  ;SeeK Shot WorK area
        MOV     CH,EMBVAL       CH ←敵の弾の総数
        MOV     BX,offset EMBWOK   BX ←敵の弾のワークエリア先頭アドレス
        MOV     DX,type bl_info    DX ←敵の弾のワークエリアの長さ
SKSLP:  ;SKSwk LooP
        MOV     AL,[BX].BL_STATUS
        OR      AL,AL
        JE      SHOTOK
        ADD     BX,DX           敵の弾のワークエリアに空きがあれば SHOTOK へ
        DEC     CH              なければ,キャリーフラグを立てる
        JNE     SKSLP
        STC
        RET
;
BDFIXW  db      Ø               BDFIX 用ワークエリア
;
SETPDD  dw      Ø               SETPD Data 用
```

```
;
SHOTOK: ;SHOT OK
        MOV     [BX].BL_STATUS,1              ┐
        MOV     DL,[SI].XZAHYOU               │
        MOV     DH,[SI].YZAHYOU               │
        ADD     DX,SETPDD                     │ 敵の弾の初期情報セット
        MOV     [BX].BL_XZAHYOU,DL            │
        MOV     [BX].BL_YZAHYOU,DH            │
        MOV     [BX].BL_HOUKOU,CL             │
        RET                                   ┘

        include LIST6-2.ASM
```

テスト6-3 テスト・プログラム(TEST6-3.ASM)

```
;
;********** TEST 6-3 **********
;
CODE    segment                        命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,SS:STACK

print   macro   string                 文字列出力マクロの定義
        LEA     DX,string              マクロパラメータのオフセットを DX へ
        MOV     AH,9                   ファンクションコール番号9…… 文字列出力
        INT     21H                    ファンクションコール
        OUT     64H,AL                 GDC にリセット・コマンドを送る
        endm                           マクロ定義終了

PTEST:  ;Program TEST
        CLD                            ストリング命令用フラグを＋方向とする
        MOV     AX,CS                  AX へコードセグメント値をセット
        MOV     DS,AX                  DS へコードセグメント値をセット
        print   CLEAR                  テキスト画面クリア
        CALL    GINIT                  グラフィック・システム初期化
        CALL    DATLD                  データ・ロード
        CALL    CLS                    画面クリア
;
        MOV     BX,26ØØH*4+226H        ┐ 自機の初期出現位置設定
        MOV     MYLOC,BX               ┘
        MOV     BX,offset MYBWOK       ┐
        MOV     CX,MYBVAL              │
IMYBLP:;Initialize MY Bullet loop      │
        MOV     [BX].MB_STATUS,Ø       │ 自機の弾のワークエリア初期化
        ADD     BX,type mb_info        │
        LOOP    IMYBLP                 ┘
```

```
;
        MOV     BX,offset EMBWOK
        MOV     DX,type bl_info
        MOV     CX,EMBVAL
IEMBLP: ;Initialize EneMy Bullet                    敵の弾のワークエリア初期化
        MOV     [BX].BL_STATUS,Ø
        ADD     BX,DX
        LOOP    IEMBLP
;
        MOV     BX,offset EMWORK
        MOV     DX,type eminfo
        MOV     CX,EMVAL
IEMLP:  ;Initialize EneMy LooP                      敵のワークエリア初期化
        MOV     [BX].STATUS,Ø
        ADD     BX,DX
        LOOP    IEMLP
        print   DSCOR                               点数表示
        CALL    SETINT                              割り込みモード設定
        CALL    IDISP                               初期画面の作成
;
MAIN:   ;MAIN loop
        CALL    SSKCK                               SPACE のチェック
        CALL    MYBMOV                              自機の弾移動
        CALL    EMBMOV                              敵の弾移動
        CALL    SCROLL                              スクロール
        CALL    MYMOVE                              自機の移動
        CALL    MYBMOV                              自機の弾移動
        CALL    EMMVAL                              敵の移動
        CALL    EMBMOV                              敵の弾移動
        CALL    PWAIT                               ウェイト
        JMP     MAIN
;
EMTTBL  dw      offset TLINE,offset TLINE           EneMy Type TaBLe
        dw      offset TLINE,offset TLINE
        dw      offset TLINE,offset TLINE
        dw      offset TLINE,offset TLINE
        dw      offset TLINE,offset TLINE
        dw      offset TLINE,offset TLINE
        dw      offset TLINE,offset TLINE
        dw      offset TLINE,offset TLINE
        dw      offset TLINE,offset TLINE
        dw      offset TLINE,offset TLINE
        dw      offset TLINE,offset TLINE
        dw      offset TLINE,offset TLINE
        dw      offset TLINE,offset TLINE
;
EMSTBL  db      Ø2H,ØØH,Ø2H,ØØH,Ø2H,ØØH             EneMy Score TaBLe
        db      Ø2H,ØØH,54 DUP (Ø)
;
EMCONT  db      Ø                                   EneMy COuNT
```

```
;
SKYP1    equ      4                                   SKY Pattern 1
EXPP1    equ      32                                  EXPlosion Pattern 1
LPS1     equ      4Ø                                  Land Pattern Small 1
LPL1     equ      44                                  Land Pattern Large 1
LDEADP   equ      38                                  Land DEAD Pattern
;
EMVAL    equ      3Ø                                  EneMy Work LENgth
;
eminfo   struc    ;EneMy INFOrmation                  ── 敵の情報の構造体定義
STATUS   db       Ø                                   敵の状態
PATTERN db        Ø                                   敵のパターン番号
XZAHYOU db        Ø                                   X座標
YZAHYOU db        Ø                                   Y座標
POINTER dw        Ø                                   ポインタ
SCORE    dw       Ø                                   スコア
HOUKOU   db       Ø                                   方向
EDUMMY   db       7 dup (Ø)                           敵情報のダミー領域
eminfo   ends                                         構造体の定義終了
;
EMWORK   db        EMVAL * type eminfo dup (Ø)        敵ワークエリア
;
bl_info  struc    ;BuLlet INFOrmation                 ── 敵の弾情報の構造体定義
BL_STATUS   db        Ø                               敵の弾の状態
BL_XZAHYOU db         Ø                               敵の弾のX座標
BL_YZAHYOU db         Ø                               敵の弾のY座標
BL_HOUKOU   db        Ø                               敵の弾の方向
bl_info  ends                                         構造体定義終了
;
EMBVAL   equ      4Ø                                  EneMy Bullet VALue
EMBWOK   db        EMBVAL * type bl_info dup (Ø)      EneMy Bullet Work area
;
mb_info  struc    ;My Bullet INFOrmation              ── 敵の弾情報の構造体定義
MB_STATUS db      Ø                                   自機の弾の状態
MB_XZAHYOU db     Ø                                   自機の弾のX座標
MB_YZAHYOU db     Ø                                   自機の弾のY座標
mb_info  ends                                         構造体定義終了
;
MYBVAL   equ      12                                  MY Bullet VALue
MYBWOK   db       MYBVAL * type mb_info dup (Ø)       MY Bullet Work area
;
REND     equ      8Ø-4                                Right END
DEND     equ      176                                 Down END
;
MYLOC    dw       Ø                                   MY LOCation
MYRST    db       Ø                                   MY ReST
SSKEY    db       ØFFH                                Set Space KEY
```

```
;
STOP    equ     Ø                               STOP
QRR     equ     1                               Direction=1
QUR     equ     2                               Direction=2
QUU     equ     3                               Direction=3
QUL     equ     4                               Direction=4
QLL     equ     5                               Direction=5
QDL     equ     6                               Direction=6
QDD     equ     7                               Direction=7
QDR     equ     8                               Direction=8
CHIJOU  equ     9                               Direction=9
;
S1      equ     1ØH                             Shot 1
S2      equ     2ØH                             Shot 2
S3      equ     3ØH                             Shot 3
;
_END    equ     8ØH                             END     command of QRL
_JUMP   equ     81H                             JUMP    command of QRL
_IFZ    equ     82H                             IFZ     command of QRL
_IFC    equ     83H                             IFC     command of QRL
_CALL   equ     84H                             CALL    command of QRL
_FETCH  equ     85H                             FETCH   command of QRL
;
TLINE   db      CHIJOU                          Test LINE
        db      CHIJOU
        db      _JUMP
        dw      offset TLINE
;
DSCOR   db      1BH,"[1;1H",1BH,"[21;33mSKY BRUISER"
MSCOR   db      1BH,"[1;5ØH",1BH,"[23;37m     得点  Ø        "
        db      1BH,"[1;72H",1BH,"[2Ø;32m残り "
MYREM   db      1BH,"[1;77H"
MYREN   db      "8 機",1BH,"[23;37m$"

lodinf  struc                                   ロード情報テーブル構造体定義
CMD     db      Ø
PASS    db      "            "
ENDSIN  db      Ø
LDADR   dw      Ø
LDSEG   dw      Ø
lodinf  ends
;
LODTBL: ;LOaD TaBLe
        lodinf <1,"SKYBRU.DAT",Ø,Ø,PTNSEG>
        lodinf <1,"MAPPAT.DAT" ,Ø,Ø,MPPSEG>
        lodinf <1,"MAPDAT2.DAT",Ø,Ø,MPDSEG>
        lodinf < >
;
        include LIST6-3.ASM
```

4. スカイ・ブルーザー… Playing Game

『会うは別れの始めなり』といいますが，マシン語ゲームの入門書として，ゼロからスタートをした本書も，事実上最後のプログラムを迎えることになりました．1章で作った最初のプログラムと比較すると，たった1冊の本でここまでくるのは，かなり無理があったような気もしますが，それを乗り越えて読破されてきたあなたの努力と根性には，心より敬意を表します．とかく他人の作ったプログラムというものは，わかりにくくそして理解するのに骨が折れるものです．それは，たとえ同じ結果を求めるプログラムでも，作った人により考え方やアルゴリズムがまったく違うからです．ちょうど碁盤の目の一角から，線に沿って対角の頂点をめざすのに，何通りもの道があるように，プログラムには絶対的な正解手順というものはありません．本書のなかにも，不満を感じるルーチンがあったと思います．逆の見方をすれば，それがマシン語をマスターした証拠でもあるわけです．はるか遠くにあったゴールが，いつのまにかこんなにも近くに来ているのです．サァ，一気に，ゴールインしてしまいましょう!!

さて，この最後のプログラムですが，マシン語プログラムというより，ほとんどが敵移動のためのQRLプログラムとなっています．また，ゲームとして最後の仕上げもしなければなりませんから，これまでのテスト・プログラムに比べてかなり長くなっています．また，パターン・データ，マップ・データ，敵データ，文字データなどの各データも必要となります．そこで，プログラムの完成と同時に，ゲームとしてもいちおう完成となるように，これらのデータのほうから先に作成していくことにしましょう．

まずは，パターン・データからですが，このゲーム〝スカイ・ブルーザー〟用に用意したマップ・エディタで使用可能なパターン数は，敵が地上用も含めて96種類，背景が96種類となっています．これ全部をデータ化するのは，QRLプログラムのことも考えると，もはやマシン語マスター用練習プログラムではなく，本物の商品作りになってしまいますから，ここでは地上敵＝3種類，空中敵＝12種類，背景＝26種類にしています．このほかに自機，爆発シーン，残骸，そして弾のパターン・データも必要ですから，これでも結構多いと感じるかもしれません．しかし，商品に近づけるという目標のためには，このくらいは最低限用意しないと面白くありません．

パターン・データの作成は巻頭口絵のパターン②，③を見ながら，パターン・エディタを使用して作ってください．

次にマップ・エディタで敵データを作成しなくてはなりません．これはAppendix 3のマップ・エディタ〝MAPEDIT〟を用いて作ります．プログラムは，紙面の都合でダンプ・リストで載せてあります．

なお，ツールの操作についてはAppendixを参考にしてください．マップ・データ

の作成が完了したら，残るはいよいよメイン・プログラムだけとなります．

メイン・プログラムでは，敵の移動2回に対し，自機の弾が5回移動するようにしています．これは，当初1対2の割合にする予定でしたが，自機の弾だけ，より速く動くようにという要望が著者のまわりで強かったため，変更したものです．

個々のQRL命令の内容については，文章で説明するより画面でその動きを見たほうがわかりやすいので，ここではとくに取り上げません．敵の動きは，自機を不死身にした上で[HOME]を押しながらジックリと観察してください．不死身にする方法は，メイン・ループ中のMYCHK後の2バイト(3箇所)を取るだけです．QRL命令によるプログラムを自分で作成してみたいという場合は，まず方眼紙に移動させたいラインを方向番号に合わせた移動距離で引いておきます．その後で，弾の発射やサブ・コマンドのことを考えながら，プログラムしていくと作りやすいと思います．

最後に，このリストで追加になったサブ・コマンドを，**表6-5**にまとめましたので参考にしてください．

また，マップ・エディタを使って自分のマップ・データを作ったら，そのファイル名をロード・ファイルテーブルに登録してからアセンブルしてください．

アセンブルが無事に済めば，長かった"スカイ・ブルーザー"のプログラムも完成に大きく近づいたことになります．

ロード・ルーチンのロード・ファイル登録テーブルにパターン・データ，マップ・データ，さらに文字データなど必要なファイルを登録すれば，すべての準備は完了です．

A>TEST6-4 ⏎

プログラムを走らせて，一発でうまく動作してくれた方は，もう我が世の春とばかりに遊んでしまってもいいでしょう．そうでない方は，再び春をめざしてツライ世界，すなわちデバッグ作業へと戻らなけれ

サブ・コマンド	内容
POSCK (POSition ChecK)	自機と敵とのY軸の差 ≧ 28H なら，キャリーフラグを立てて戻る
SETSI (SET IX register)	(SI＋次の1バイト値)に，その次の1バイト値を入れる
CTSI (Count IX)	(SI＋次の1バイト値)を－1し， ゼロでなければキャリーフラグを立てて戻る
INTSC (INiTialize Sky C)	空中敵－C専用のローカル・ワーク値設定 ……ワークエリア内容についてはプログラム参照
REVIVE (REVIVE enemy)	空中敵－C専用の移動ルーチン ……弾に当っても5回までは不死身で，SHOTADを行う
使用例 _CALL　SETSI　<offset><value> _IFC　CTSI　<offset><NP>	 (SI＋<offset>) ← <value> (SI＋<offset>)を－1し0であったら<NP>へジャンプ

表6-5 サブ・コマンド

ばなりません．といっても，完動のリストがいちおうあるわけですから，春が来るのはもう時間の問題です．頑張ってください．

Quasi（グワシ！……ではありません，クワージです）大作の〝スカイ・ブルーザー〞，いかがでしたか．コンストラクション機能をフルに使えば，かなり長く楽しめる反面，商品にするには今0.3歩の壁を感じる方もいるかもしれません．たとえば，タイトルやデモ画面，ゲームにおける奥の深さ，意外性，パターンの総数，そしてサウンド……と数えたらキリがありません．しかし，それらは世に出ている商品の価値を保つためにも，またあなた自身の創造性を養うためにも，「なくてよかった」と思って欲しいのです．本書にあるすべての作品に対し，大いに不満を感じてもらうことができたならば，正に本書の目的は100%達成できたといっても過言ではないでしょう．要は，その不満をあなた自身の作品にぶつければいいのです．そこには，単なるモノマネではない確固たる技術の進歩があるからです．どうか，あなた自身の工夫，改良によるスバラシイ作品を作られることを，心より期待します．間違っても，筆者への怒りの投書などという形で，不満の表現をすることだけはしないでください．ネッ!!

リスト6-4 スカイ・ブルーザーの仕上げ(TEST 6-4.ASM)

```
;
;*********** TEST 6-4 ***********
;
CODE    segment                          命令の置かれているセグメントの始まり

        assume CS:CODE,DS:CODE,SS:STACK
print   macro   string                   文字列出力マクロの定義
        LEA     DX,string                マクロパラメータのオフセットをDXへ
        MOV     AH,9                     ファンクションコール番号9……文字列出力
        INT     21H                      ファンクションコール
        OUT     64H,AL                   GDCにリセット・コマンドを送る
        endm                             マクロ定義終了

PTEST:  ;Program TEST
        CLD                              ストリング命令用フラグを＋方向とする
        MOV     AX,CS                    AXへコードセグメント値をセット
        MOV     DS,AX                    DSへコードセグメント値をセット
        print   CLEAR                    テキスト画面クリア
        CALL    GINIT                    グラフィック・システム初期化
        CALL    DATLD                    データ・ロード
        CALL    CLS                      画面クリア
;
        IN      AL,73H                  }
        AND     AL,7                    }乱数の初期値セット
        MOV     RNDWOK,AL               }
        MOV     BX,offset DTITLE        }
        MOV     CX,1614H                }“SKY BRUISER”の表示
        CALL    MSGPRN                  }
        MOV     BX,offset PRESS         }
        MOV     CX,3F18H                }「PRESS RETURN KEY」の表示
        CALL    MSGPRN                  }
        CALL    SETINT                   割り込みモードの設定
TLOOP:  ;Test LOOP
        XOR     AX,AX                    AX←0
        MOV     KPSTOP,AL                KPSTOP初期化
        MOV     AL,RETKEY               }
        AND     AL,AL                   }RETURNが押されていなければTLOOPへ
        JE      TLOOP                   }
        XOR     AX,AX                    AX←0
        MOV     RETKEY,AL                RETKEY初期化
```

```
        MOV     AL,KPHELP
        XOR     AL,1
        JNE     TØ
        MOV     HELP,AL
        MOV     KPHELP,AL
        JMP     T1
TØ:     ;Test Ø
        MOV     HELP,AL
T1:     ;Test 1
        print   CLEAR
        print   DSCOR
        MOV     IDOZAH,SCLPTC*2
        MOV     EMYSIN,Ø
        MOV     EMYNO,Ø
        MOV     ENEMYX,Ø
        MOV     MPBADR,8ØH-2*4
        MOV     MAPEND,Ø
        MOV     MAPADR,Ø
        MOV     KPMAPN,Ø
        MOV     SCLAD1,4ØØØH+SCLPTC
        MOV     SCLNO1,Ø
        MOV     SCLNO2,Ø
        CALL    CLS
        CALL    IDISP
        MOV     AL,8
        MOV     MYRST,AL
        OR      AL,3ØH
        MOV     MYREN,AL
        print   MSCOR
        MOV     BX,Ø
        MOV     word ptr SCORE2,BX
        MOV     word ptr SCORE2+1,BX
;
        MOV     BX,offset MYBWOK
        MOV     CX,MYBVAL
IMYBLP:;Initialize MY Bullet loop
        MOV     [BX].MB_STATUS,Ø
        ADD     BX,type mb_info
        LOOP    IMYBLP
;
        MOV     BX,offset EMBWOK
        MOV     DX,type bl_info
        MOV     CX,EMBVAL
IEMBLP: ;Initialize EneMy Bullet
        MOV     [BX].BL_STATUS,Ø
        ADD     BX,DX
        LOOP    IEMBLP
```

HELP モード・セット
テキスト画面クリア
点数の表示
スクロール用ワークエリアの初期化
画面クリア
初期画面セット
自機残数の設定
スコアの初期化
自機の弾のワークエリア初期化
敵の弾のワークエリア初期化

```
;
          MOV       BX,offset EMWORK
          MOV       DX,type eminfo
          MOV       CX,EMVAL
IEMLP:    ;Initialize EneMy LooP                   敵のワークエリア初期化
          MOV       [BX].STATUS,Ø
          ADD       BX,DX
          LOOP      IEMLP
BEGIN:    ;BEGIN
          MOV       BX,26ØØH*4+226H                自機の初期出現位置設定
          MOV       MYLOC,BX
          MOV       CH,1ØH                         スタート時のフラッシュ回数
BEGLP:    ;BEGin LooP
          PUSH      CX                             ループ数退避
          CALL      SSKCK                          SPACEのチェック
          CALL      MYBMOV                         自機の弾移動
          CALL      EMBMOV                         敵の弾移動
          CALL      EMMVAL                         敵の移動
          CALL      SCROLL                         スクロール
          CALL      MYMOVE                         自機の移動
          CALL      SSKCK                          SPACEのチェック
          CALL      MYBMOV                         自機の弾移動
          CALL      MYBMOV                         自機の弾移動
          CALL      PWAIT                          ウェイト
          CALL      SSKCK                          SPACEのチェック
          CALL      MYBMOV                         自機の弾移動
          CALL      EMBMOV                         敵の弾移動
          CALL      EMMVAL                         敵の移動
          CALL      SCROLL                         スクロール
          CALL      MYMOVE                         自機の移動
          MOV       BX,42ØH                        自機の消去
          MOV       CX,MYLOC
          CALL      CLPTXY
          CALL      SSKCK                          SPACEのチェック
          CALL      MYBMOV                         自機の弾移動
          CALL      EMBMOV                         敵の弾移動
          CALL      PWAIT                          ウェイト
          POP       CX                             ループ数復元
          DEC       CH                             CH回ループ
          JNE       BEGLP
;
MAIN:     ;MAIN loop
          MOV       KPSTOP,Ø
          MOV       RETKEY,Ø
          CALL      SSKCK
          CALL      MYBMOV
          CALL      EMBMOV
          CALL      EMMVAL
          CALL      SCROLL
          CALL      MYCHK
          JB        MYDEAD
```

```
        CALL    MYMOVE
        CALL    SSKCK
        CALL    MYBMOV
        CALL    MYBMOV
        CALL    PWAIT
        CALL    SSKCK
        CALL    MYBMOV
        CALL    EMBMOV
        CALL    EMMVAL
        CALL    SCROLL
        CALL    MYCHK
        JB      MYDEAD
        CALL    MYMOVE
        CALL    SSKCK
        CALL    MYBMOV
        CALL    EMBMOV
        CALL    MYCHK
        JB      MYDEAD
        CALL    PWAIT
M1:     ;Main 1
        MOV     AL,KEYDAT
        TEST    AL,10H
        JE      MAIN
        MOV     CX,2000H
M2:     ;Main 2
        LOOP    M2
        JMP     M1
;
MYDEAD: ;MY DEAD
        MOV     CX,8
        PUSH    CX
        JMP     MDLP1
MDLP:   ;MyDead LooP
        PUSH    CX
        CALL    MYBMOV
        CALL    EMBMOV
        CALL    EMMVAL
        CALL    SCROLL
MDLP1:  ;MyDead LooP 1
        CALL    MYBMOV
        MOV     CX,MYLOC
        ADD     CH,16
        MOV     BX,400H+SCLDOT
        CALL    CLPTXY
        MOV     CX,MYLOC
        MOV     AL,EXPP1
        CALL    DISP
        CALL    MYBMOV
        CALL    PWAIT
        CALL    MYBMOV
        CALL    EMBMOV
```

(コメント)
- SPACE のチェック …4回
- 自機の弾移動 …5回
- 敵の弾移動 …4回
- 敵の移動 …2回
- 衝突の判定 …3回
- スクロール …1回
- 自機の移動 …2回
- ウェイト …2回

- AL←キーの情報
- HOME が押されたか?
- 押されていなければMAINへ
- ウェイト
- M1へ

- CX←爆発時のループ回数
- レジスタ保存
- MDLP1へ
- レジスタ保存

- 自機の弾移動 …3回
- 敵の弾移動 …2回
- 敵の移動 …1回
- スクロール …1回
- ビープ音＋爆発パターン1の表示 …1回
- ウェイト

```
        CALL    EMMVAL
        CALL    SCROLL
        CALL    EMBMOV
        CALL    MYBMOV
        MOV     CX,MYLOC
        ADD     CH,16
        MOV     BX,4ØØH+SCLDOT
        CALL    CLPTXY
        MOV     CX,MYLOC
        MOV     AL,EXPP2
        CALL    DISP
        CALL    PWAIT
        POP     CX
        LOOP    MDLP
        MOV     BX,42ØH
        MOV     CX,MYLOC
        CALL    CLPTXY
        MOV     BX,offset MYRST
        DEC     byte ptr [BX]
        MOV     AL,[BX]
        OR      AL,3ØH
        MOV     MYREN,AL
        print   MYREM
        MOV     AL,[BX]
        AND     AL,AL
        JE      NBEGIN
        JMP     BEGIN
NBEGIN: ;Not BEGIN
        MOV     BX,offset GOVER
        MOV     CX,361FH
        CALL    MSGPRN
        MOV     BX,offset PRESS
        MOV     CX,5F18H
        CALL    MSGPRN
        MOV     byte ptr KPHELP,Ø
        JMP     TLOOP
;
PRESS   db      'PRESS RETURN KEY',Ø

DSCORE  db      'SCORE',Ø

DTITLE  db      'S K Y  B R U I S E R',Ø

GOVER   db      'GAME OVER',Ø
;
EMTTBL  dw      LAND1,LAND1,LAND2,LAND3,SKY1 ,SKY2 ,SKY3 ,SKY4
        dw      SKY5 ,SKY6 ,SKY7 ,SKY8 ,SKY9 ,SKYA ,SKYB ,SKYC
        dw      SKY1 ,SKY1 ,SKY1 ,SKY1 ,SKY1 ,SKY1 ,SKY1 ,SKY1
        dw      SKY1 ,SKY1 ,SKY1
```

（コメント）
- 自機の弾移動　…２回
- 敵の弾移動　…２回
- 敵の移動　…１回
- スクロール　…１回
- ビープ音＋爆発パターン２の表示 …１回
- ウェイト
- レジスタ復元
- CX 回ループ
- 爆発した自機の消去
- 自機の残数を
- 自機の残数表示
- 自機の残数が０でなければ BEGIN へ
- 「GAME OVER」表示
- 「PRESS RETURN KEY」表示
- TLOOP へ

```
EMSTBL  dw      3,  3,  6,  9,  1,  2,  3,  4
        dw      5,  6,  7,  8,  9,1ØH,11H,12H
        dw      Ø,  Ø,  Ø,  Ø,  Ø,  Ø,  Ø,  Ø
        dw      Ø,  Ø,  Ø
;
SKYPC   equ     15
;
LAND1   db      _CALL                                  地上敵１
        dw      LSHOT1
        db      CHIJOU
        db      CHIJOU
        db      _JUMP
        dw      LAND1
;
LAND2   db      _CALL                                  地上敵２
        dw      LSHOT2
        db      CHIJOU
        db      CHIJOU
        db      _JUMP
        dw      LAND2
;
LAND3   db      _CALL                                  地上敵３
        dw      LSHOT3
        db      CHIJOU
        db      CHIJOU
        db      _JUMP
        dw      LAND3
;
SKY1    db      QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD        空中敵１
        db      _IFC
        dw      WHLR,S1R
        db      QDL+S1,QDD,QDD+S1,QDL,QDD+S1,QDL,QDD+S1,QDL
        db      _IFC
        dw      WHLR,SKY1
        db      QDL+S1,QDD,QDL,QDL+S1,QLL,QDL,QDL+S1,QDL,QLL
        db      QLL+S1,QLL,QLL,QLL+S1
        db      _IFC
        dw      WHLR,S1L3
        db      QDL+S1,QDD,QDL+S1,QDD,QDD+S1,QDL,QDD+S1,QDD
        db      QDL,QDD,QDD,QDD,QDD,QDD
        db      _IFC
        dw      WHLR,S1L1

S1L     db      QDL,QDD+S1
        db      _JUMP
        dw      S1L

S1L1    db      QDR,QDD,QDD+S1

S1L2    db      QDD,QDD+S1,QDD,QDL+S1
        db      _JUMP
        dw      S1L3
```

```
S1L3    db      QLL,QLL+S1,QLL,QUL+S1
        db      _JUMP
        dw      S1L3

S1R     db      QDR,QDD+S1,QDD,QDR+S1,QDD,QDR+S1,QDD,QDR+S1
        db      _IFC
        dw      WHLR,S1R1
        db      _JUMP
        dw      SKY1

S1R1    db      QDR,QDD+S1,QDR,QDR+S1,QRR,QDR+S1,QDR,QDR+S1
        db      QRR,QRR,QRR,QRR,QRR
        db      _IFC
        dw      WHLR,S1R3

S1R2    db      QRR,QRR+S1,QRR,QUR+S1
        db      _JUMP
        dw      S1R2

S1R3    db      QDL,QDD,QDL+S1,QDD,QDD,QDL+S1,QDD,QDD,QDL+S1
        db      QDD,QDD,QDD+S1,QDD,QDD
        db      _IFC
        dw      WHLR,S1R5
        db      QDL,QDD,QDD+S1

S1R4    db      QDD,QDD+S1,QDD,QDR+S1
        dw      S1R4

S1R5    db      QDR,QDD+S1
        db      _JUMP
        dw      S1L

;
SKY2    db      QDD,QDD,QDD,QDL,QDD,QDL,QDL,QDD,QDD,QDL,QDL,QDD      空中敵2
        db      _IFC
        dw      WHLR,S2R
        db      QDD,QDL,QDL,QDD,QDL,QDL,QDL,QDL,QDL,QDL,QDL,QDL,QDL

S2R     db      QUU+S3,QUR,QUU,QUR,QUR,QUU,QUR,QUU,QUR,QUU,QUR,QUU
        db      QUR,QUR,QUR,QUR,QUU,QUR,QUU,QUR,QUR,QUR,QUR,QUR,QUR
        db      _IFC
        dw      WHLR,S2R1
        db      _JUMP
        dw      S2L

S2R1    db      QUR,QUR,QUR,QUR,QUR,QUR,QUR,QRR,QUR,QUR,QUR,QRR,QUR
        db      QUR,QUR,QRR,QUR,QUR,QRR

S2L     db      QLL+S3,QLL,QLL,QLL,QLL,QDL,QLL,QLL,QDL,QDL,QDL,QDL
        db      QDL,QDL,QDL,QDL
```

```
        db      _IFC
        dw      WHLR,S2R2
        db      QDL,QDL,QDL,QDL,QDL,QDD,QDL,QDL,QDL,QDD,QDL,QDL,QDD,QDL

S2R2    db      QUU+S3,QUR,QUU,QUR,QUU,QUR,QUR,QUU,QUR,QUU,QUR,QUR
        db      QUU,QUR,QUR,QUR,QUR,QUU,QUR,QUR
        db      _IFC
        dw      WHLR,S2R3
        db      _JUMP
        dw      S2L1

S2R3    db      QUR,QUR,QUR,QUR,QUR,QUR,QUR,QRR,QUR,QUR,QUR,QRR,QUR,QUR

S2L1    db      QDL+S3,QDL,QDL,QDD,QDL,QDL,QDL,QDD,QDL,QDD,QDL,QDD,QDL
        db      _IFC
        dw      WHLR,S2R4
        db      QDD,QDL,QDD,QDD,QDL,QDD,QDD,QDL,QDD,QDD,QDL,QDD

S2R4    db      QUR+S3,QUR,QUR,QUR,QUR,QUR,QUR,QUR,QUR,QUR,QRR,QUR,QUR
        db      _IFC
        dw      WHLR,S2R5
        db      _JUMP
        dw      S2L2

S2R5    db      QUR,QRR,QUR,QUR,QUR,QRR,QUR,QUR,QRR,QUR,QUR,QRR,QUR,QRR,QUR

S2L2    db      QDL+S3,QDL

S2L3    db      QDL,QDL,QDD,QDD
        db      _JUMP
        dw      S2L3
;
SKY3    db      QDD                                     空中敵 3
        db      _IFC
        dw      POSCK,SKY3
        db      _IFC
        dw      WHLR,S3R
        db      _IFC
        dw      WHLR,S3M

S3L     db      QDD,QDD,QDD,QDL,QDD,QDL,QDL,QLL,QLL,QLL,QLL
        db      _CALL
        dw      SHOTAD
        db      QUL,QLL
        db      _CALL
        dw      SHOTAD
        db      QUL,QUL,QUU,QUU,QUL,QUU,QUL,QUU,QUU,QUU,QUU,QUU,QUL

S3U     db      QUU
        db      _JUMP
        dw      S3U
```

```
S3M     db      QDD,QDD,QDD,QDD,QDD,QDD,QDD,STOP,STOP,STOP,STOP
        db      _CALL
        dw      SHOTAD
        db      QUU,STOP
        db      _CALL
        dw      SHOTAD
        db      QUU,STOP,QUU
        db      _JUMP
        dw      S3U

S3R     db      QDD,QDD,QDD,QDR,QDD,QDR,QDR,QRR,QRR,QRR,QRR
        db      _CALL
        dw      SHOTAD
        db      QUR,QRR
        db      _CALL
        dw      SHOTAD
        db      QUR,QUR,QUU,QUU,QUR,QUU,QUR,QUU,QUU,QUU,QUU,QUU,QUR
        db      _JUMP
        dw      S3U
;
POSCK:  ;POSition Check
        MOV     AL,byte ptr MYLOC+1
        SUB     AL,[SI+3]
        CMP     AL,28H
        CMC
        RET
;
;
SKY4    db      QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD    空中敵4
        db      QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD
        db      QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD
        db      _IFC
        dw      WHLR,S4R

S4L     db      QDL

S4L1    db      QLL,QUL
        db      _IFC
        dw      WHLR,S4L2
        db      _JUMP
        dw      S4L1

S4L2    db      QUL+S2,QUU+S2,QUL+S2,QUU+S2,QUL+S2,QUU+S2
        db      QUU+S2,QUL+S2,QUU+S2,QUU+S2,QUU+S2,QUU+S2
        db      QUU+S2,QUU+S2,QUU+S2,QUU+S2,QUU+S1,QUU+S1
        db      QUU+S1,QUU+S1,QUU+S1,QUU+S1,QUU+S1,QUU+S1
        db      QUU+S1,QUU+S1,QUU+S1,QUU+S1,QUU+S1,QUU+S1
        db      QUU+S1,QUU+S1,QUU,QUU,QUR,QUU,QUU,QUU,QUU,QUU
        db      QUU,QUR
        db      QUR,QUU,QUU,QUU,QUU,QUU,QUU,QUU,QUU,QUU,QUU,QUU
```

```
S4L3    db      QUU
        db      _JUMP
        dw      S4L3

S4R     db      QDR
S4R1    db      QRR,QUR
        db      _IFC
        dw      WHLR,S4R1

S4R2    db      QUL+S2,QUU+S2,QUR+S2,QUU+S2,QUR+S2,QUU+S2
        db      QUU+S2,QUU+S2,QUU+S2,QUU+S2,QUU+S2,QUU+S2
        db      QUU+S2,QUU+S2,QUU+S2,QUU+S2,QUU+S1,QUU+S1
        db      QUU+S1,QUU+S1,QUU+S1,QUU+S1,QUU+S1,QUU+S1
        db      QUU+S1,QUU+S1,QUU+S1,QUU+S1,QUU+S1,QUU+S1
        db      QUU+S1,QUU+S1,QUU,QUU,QUL,QUU,QUU,QUU,QUU,QUU
        db      QUL,QUU,QUU,QUU,QUU,QUU,QUU,QUU,QUU,QUU,QUU,QUU
        db      QUU,QUL
        db      _JUMP
        dw      S4L3
;
;
SKY5    db      QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD      空中敵 5
        db      QDD,QDD,QDD,QDD
        db      _IFC
        dw      WHDU,S51
        db      QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD,QDD
        db      QDD,QDD,QDD,QDD

S51     db      QDR,QDD,QDD,QDR,QDR,QDR,QRR

S52     db      QUR,QRR
        db      _IFC
        dw      WHLR,S52
        db      QUR,QRR,QUR,QUR,QUR,QRR,QUR,QUU,QUR,QUU,QUR,QUU
        db      QUU,QUR,QUU,QUU,QUU,QUU,QUR,QUU,QUU,QUU,QUU,QUU
        db      QUL,QUU,QUU,QUU,QUL,QUL,QUL,QUL,QUL,QUL,QLL,QUL
        db      QLL,QLL,QDL,QDD,QDL,QDD,QDD,QDD,QDD,QDR,QDD,QRR
        db      QRR,QUR,QRR,QUR,QUR,QUR,QUR,QUU,QUU,QUU,QUU,QUR
        db      QUU,QUU,QUU,QUU,QUU,QUL,QUU,QUU,QUL,QUL,QUL,QLL
        db      QUL,QDL,QDL,QDL,QDD,QDD,QDD,QDR,QRR,QRR,QUR,QUR
        db      QUR,QUU,QUU,QUU,QUU,QUU,QUU,QUU,QUL,QUL,QLL,QDL
        db      QDD,QDR,QRR,QUR,QUU,QUU,QUL,QLL
        db      STOP
        db      _CALL
        dw      SHOTAD
        db      STOP
        db      _CALL
        dw      SHOTAD
        db      STOP
        db      _CALL
        dw      SHOTAD
        db      _END
```

```
;
SKY6    db      QDD,QDD+S1          空中敵6
        db      _IFC
        dw      WHDU,S6LR
        db      _JUMP
        dw      SKY6

S6LR    db      _IFC
        dw      WHLR,S6R
        db      _IFZ
        dw      WHLR,SKY6

S6L     db      QDD,QDD,QDL

S6L1    db      QLL+S1,QLL
        db      _JUMP
        dw      S6L1
S6R     db      QDD,QDD,QDR

S6R1    db      QRR+S1,QRR
        db      _JUMP
        dw      S6R1
;
SKY7    db      _CALL               空中敵7
        dw      SETSI
        db      1Ø,8

S7LP    db      _CALL
        dw      SETSI
        db      11,16
        db      _FETCH
        dw      DIRME

S7LP1   db      _CALL
        dw      ESHOT1
        db      _FETCH
        dw      SAMDIR
        db      _IFC
        dw      CTSI
        db      11
        dw      S7LP1
        db      _IFC
        dw      CTSI
        db      1Ø
        dw      S7LP

S7LP2   db      _FETCH
        dw      SAMDIR
        db      _JUMP
        dw      S7LP2
```

```
;
SETSI:  ;SET SI
        MOV     BP,SP                        BP←SP
        MOV     BX,[BP+2]                    BX←コマンド・ポインタ
        ADD     word ptr [BP+2],2            コマンド・ポインタ更新
        INC     BX
        MOV     BX,[BX]
        XOR     AL,AL                        [SI+*]にデータをセット
        XCHG    AL,BH
        MOV     [SI+BX],AL
        POP     BX                           BX←リターン・アドレス
        JMP     BX                           リターン
;
CTSI:   ;CounT SI
        MOV     BP,SP                        BP←SP
        INC     word ptr [BP+2]
        MOV     BX,[BP+2]
        MOV     BL,[BX]
        XOR     BH,BH                        [SI+*]の値を−1し0でなければ
        DEC     byte ptr [SI+BX]             キャリー・フラグを立てる,0のときはCX2へ
        JE      CT2
        STC
        POP     BX
        JMP     BX
CT2:    ;CTSI 2
        OR      AX,AX                        キャリー・フラグのリセット
        POP     BX                           BX←リターン・アドレス
        JMP     BX                           リターン
;

SKY8    db      QDD+S1,QDL+S1,QDD+S1,QDD+S1,QDL+S1,QDD+S1    空中敵8
        db      QDL+S1,QDD+S1,QDD+S1,QDL+S1,QDD+S1,QDL+S1
        db      QDD+S1,QDD+S1,QDD+S1,QDL+S1,QDD+S1,QDD+S1
        db      QDL+S1,QDD+S1,QDD+S1,QDD+S1,QDL+S1,QDD+S1
        db      QDD,QDD,QDD,QDL,QDD,QDD,QDD,QDD
        db      QDL,QDD,QDD,QDD,QDD,QDL,QDD,QDD
        db      QDD,QDD
        db      _IFC
        dw      WHLR,S8R

S8L     db      _CALL
        dw      SETSI
        db      10,32

S8L1    db      QUU+S1
        db      _IFC
        dw      CTSI
        db      10
        dw      S8L1
        db      QUL,QUU,QUU,QUU,QUU,QUU,QUU,QUU
        db      QUU,QUU,QUU,QUU,QUU,QUU,QUL,QUU
```

```
        db      QUU,QUU,QUU,QUU,QUU,QUU,QUU,QUU
        db      QUU,QUU,QUU,QUU,QUL,QUU,QUU,QUU
        db      QUU,QUU,QUU,QUU,QUU,QUU,QUR,QUU
        db      QUU,QUU,QUU,QUU,QUR,QUU,QUU,QUU
        db      QUU,QUU,QUU,QUU,QDD,QDL,QDD,QDD
        db      QDL,QDD,QDL,QDD
        db      QDD+S1,QDL+S1,QDD+S1,QDL+S1,QDD+S1,QDD+S1
        db      QDD+S1,QDL+S1,QDD+S1,QDD+S1,QDL+S1,QDD+S1
        db      QDD+S1,QDD+S1,QDL+S1,QDD+S1,QDD+S1,QDD
        db      QDD+S1,QDL,QDD+S1,QDD,QDD+S1,QDD
        db      QDL,QDD,QDD,QDD,QDD,QDL,QDD,QDD
        db      QDD,QDD,QDD,QDD,QDD,QDD
        db      _JUMP
        dw      S8L
S8R     db      QUU+S2,QUR+S2,QUU+S2,QUU+S2
        db      QUU+S2,QUU+S2,QUU+S2,QUU+S2
        db      QUR,QUU,QUU,QUU,QUU,QUR,QUU,QUU
        db      QUU,QUU,QUU,QUU,QUU,QUR,QUU,QUU
        db      QUU,QUU,QUR,QUU,QUU,QUU,QUU,QUU
        db      QUU,QUU,QUU,QUU,QUR,QUU,QUU,QUU
        db      QUU,QUU,QUU,QUU,QUR,QUU,QUU,QUR
        db      QUU,QUU,QUU,QUU,QUR,QUU,QUU,QUU
        db      QUR,QUU,QUU,QUU,QUR,QUU,QUU,QUR
        db      QUU,QUU,QUR,QUU,QUU,QUU,QUR,QUU
        db      QUR,QUU,QUU,QUR,QUU,QUR,QUU,QUU
        db      QUR,QUU,QUR,QUU,QDD,QDL,QDD,QDD
        db      QDL,QDD,QDL,QDD,QDD+S1,QDL+S1
        db      QDD+S1,QDL+S1,QDD+S1,QDD+S1
        db      QDD+S1,QDL+S1,QDD+S1,QDD,QDL+S1
        db      QDD,QDD+S1,QDD,QDL+S1,QDD,QDD+S1
        db      QDD,QDD,QDL,QDD+S1,QDD,QDD,QDD
        db      QDL+S1,QDD,QDD,QDD,QDD,QDL,QDD
        db      QDD,QDD+S1,QDD,QDD,QDD,QDD,QDD
        db      _JUMP
        dw      S8R
;
SKY9    db      _CALL                   空中敵9
        dw      SETSI
        db      10,18
S9LP    db      _FETCH
        dw      SWINGD
        db      _CALL
        dw      SETSI
        db      11,8
S9LP1   db      _CALL
        dw      ESHOT1
```

```
        db      _FETCH
        dw      SAMDIR
        db      _IFC
        dw      CTSI
        db      11
        dw      S9LP1
        db      _IFC
        dw      CTSI
        db      1Ø
        dw      S9LP

S9LP2   db      _FETCH
        dw      SAMDIR
        db      _JUMP
        dw      S9LP2
;
SKYA    db      QLL,QDL,QDD,QDR,QDD,QDL,QDL,QDD       空中敵 A
        db      QDD,QDR,QDD,QDD,QDL,QDL,QDD
        db      _IFC
        dw      WHLR,SARØ
        db      _IFZ
        dw      WHLR,SARØ

SALØ    db      QLL,QDL,QDD,QLL,QLL,QUL,QLL
        db      _IFC
        dw      WHDU,SAL7

SAL1    db      QDL+S1,QDD+S1,QDD+S1,QDL+S1,QLL+S1,QUL+S1
        db      QUL+S1,QUL+S1,QDL+S1,QDD+S1,QDD+S1,QDD+S1
        db      QDL+S1,QLL+S1,QUL+S1,QLL+S1,QUL+S1,QDD+S1
        db      QDD+S1,QDD+S1,QDD+S1,QDL+S1
        db      _IFC
        dw      WHDU,SAL4

SAL2    db      QDL,QUL,QLL,QUL,QUL,QDD,QDD,QDD
        db      QDD,QDL,QUL,QLL,QUL,QLL,QDD,QDD

SAL3    db      QDD,QLL
        db      _JUMP
        dw      SAL3

SAL4    db      QDR,QDD,QDL,QRR,QRR,QDD,QDD,QRR
        db      QRR,QRR,QDD,QDD

SAL5    db      QDD,QDL,QDD,QDL,QDD

SAL6    db      QDL,QDD,QDD
        db      _JUMP
        dw      SAL6
```

```
SAL7    db      QUL+S1,QUL+S1,QUU+S1,QUU+S1,QUL+S1,QDL+S1
        db      QLL+S1,QUL+S1,QUL+S1,QUU+S1,QUU+S1,QUL+S1
        db      QUU+S1,QUL+S1,QUL+S1,QLL+S1,QUL+S1,QUU+S1
        db      QUU+S1,QUU+S1,QUU+S1,QUU+S1,QUU+S1,QLL+S1
        db      QLL+S1,QUL+S1,QUL+S1,QUL+S1,QUU+S1,QUU+S1
        db      QUL+S2,QUU+S1,QUU+S1,QUU+S1,QUU+S1,QUU+S1
        db      QUU+S1,QUU+S1,QLL+S1,QLL+S1,QLL+S1,QUL+S1
        db      QUL+S1,QUU+S1,QUR+S1,QUU+S1,QUU+S1,QUR+S1
        db      QUU+S1,QUU+S1,QLL+S1,QLL+S1
SAL8    db      QUL+S1,QLL+S1,QUL+S1
        db      _JUMP
        dw      SAL8
SARØ    db      QRR,QRR,QDD,QDL,QDD,QDL,QDD,QLL
        db      QLL,QLL
        db      _IFC
        dw      WHLR,SAR5
SAR1    db      QDL+S1,QDD+S1,QDL+S1,QDL+S1,QDD+S1,QDD+S1
        db      QDR+S1,QDD+S1,QDD+S1,QDD+S1,QDD+S1,QDD+S1
        db      _IFC
        dw      WHLR,SAR3
SAR2    db      _IFZ
        dw      WHLR,SAR3
        db      QLL,QUL,QDL,QDD,QDL,QLL,QUL,QLL
        db      QUL,QLL,QDL,QDL,QLL,QUL,QUL
        db      _JUMP
        dw      SAL5
SAR3    db      QDD,QDR
SAR4    db      QDD
        db      _JUMP
        dw      SAR4
SAR5    db      QRR+S1,QRR+S1,QUR+S1,QUR+S1,QRR+S1,QDR+S1
        db      QDD+S1,QRR+S1,QRR+S1,QUR+S1,QRR+S1,QRR+S1
        db      QDR+S1,QDR+S1,QDR+S1,QDR+S1
SAR6    db      QRR+S1
        db      _JUMP
        dw      SAR6
;
SKYB    db      _CALL                    空中敵B
        dw      SHOTAD
        db      STOP,STOP,STOP,STOP,STOP,STOP,STOP,STOP
        db      STOP,STOP,STOP,STOP,STOP,STOP,STOP,STOP
        db      _JUMP
        dw      SKYB
```

```
;
SKYC    db      _CALL                                           空中敵C
        dw      INTSC
SCLP    db      _CALL
        dw      REVIVE
        db      _JUMP
        dw      SCLP

SC1     db      QDL,QDL,QDL,QDD,QDD,QDD,QDL,QDL,QDL

SC2     db      QDD,QDR,QDR,QDR,QRR,QRR,QUR,QRR,QUR,QRR,QUR,QUR
        db      QUR,QRR,QUR,QUR,QRR,QUR,QUR,QUR,QUR,QUR,QRR,QUR
        db      QUR,QUR,QUR,QUR,QUR,QUR,QUU,QUR,QUR,QUU,QUR,QUU
        db      QUU,QUU,QUU,QUL,QUU,QUL,QLL,QUL,QLL,QUL,QLL,QUL
        db      QLL,QUL,QLL,QLL,QUL,QLL,QLL,QLL,QUL,QLL,QLL,QLL
        db      QLL,QLL,QLL,QLL,QLL,QLL,QDL,QLL,QDL,QDL,QDL,QDL
        db      QDD,QDL,QDD
        db      _JUMP
        dw      SC2
;
INTSC:  ;INiTialiaze Sky C
        MOV     byte ptr [SI+1Ø],4              [SI+10]←4 …… 復活の回数
        MOV     BX,offset SC1                   } コマンド・ポインタを設定する
        MOV     [SI+12],BX
        MOV     byte ptr [SI+14],SKYPC          空中敵12番のパターン番号
        RET

REVIVE: ;REVIVE enemy
        POP     DX                              DX←リターンアドレス(使用しない)
        MOV     BX,[SI+12]                      } コマンド・ポインタを移す
        MOV     [SI+4],BX
        MOV     AL,14                           } 敵移動ルーチンの最後で,表示をする際
        MOV     byte ptr EMDISP+13,AL             [SI+14]の番号を表示するようにする
        CALL    ENEMY                           敵の処理
        MOV     AL,1                            } 通常処理に戻す
        MOV     byte ptr EMDISP+13,AL
        MOV     AL,[SI+Ø]                       } 画面から外に出た場合はRVENDへ
        OR      AL,AL
        JE      RVEND
        INC     AL                              [SI+0]=FFHであれば,
        JE      RV1                             復活処理はしない
        MOV     byte ptr [SI+Ø],ØFFH            } 復活処理
        MOV     byte ptr [SI+1],SKYPC             フラグ・セット,パターン番号セット
        CALL    SHOTAD                            弾の発射
        DEC     byte ptr [SI+1Ø]                  復活回数を-1し,0ならRVENDへ
        JE      RVEND                             0でなければRV1へ
        JMP     RV1
RVEND:  ;ReVive END
        POP     BX                              BX←コマンドのポインタ
        RET                                     EMMOVE終了のリターン
```

```
RV1:    ;ReVive 1
        MOV     BX,[SI+4]               } 実際のコマンド・ポインタを[SI+12]に戻す
        MOV     [SI+12],BX              }
        POP     BX                        BX←コマンド・ポインタ
        INC     BX                      } [SI+4]には,つねにREVIVEが実行される
        MOV     [SI+4],BX               } ような状態のコマンド・ポインタとなる
        RET                             } (SCLP内の_JUMPとなっている)
;
EMCONT  db      Ø                         EneMy COuNT
SKYP1   equ     4                         SKY Pattern 1
EXPP1   equ     32                        EXPlosion Pattern 1
EXPP2   equ     33                        EXPlosion Pattern 2
LPS1    equ     4Ø                        Land Pattern Small 1
LPL1    equ     44                        Land Pattern Large 1
LDEADP  equ     38                        Land DEAD Pattern
;
EMVAL   equ     3Ø                        EneMy Work LENgth
;
eminfo  struc   ;EneMy INFOrmation        ── 敵の情報の構造体定義
STATUS  db      Ø                         敵の状態
PATTERN db      Ø                         敵のパターン番号
XZAHYOU db      Ø                         X座標
YZAHYOU db      Ø                         Y座標
POINTER dw      Ø                         ポインタ
SCORE   dw      Ø                         スコア
HOUKOU  db      Ø                         方向
EDUMMY  db      7 dup (Ø)                 敵情報のダミー領域
eminfo  ends                              構造体の定義終了
;
EMWORK  db      EMVAL * type eminfo dup (Ø)   敵ワークエリア
;
bl_info struc   ;BuLlet INFOrmation       ── 敵の弾情報の構造体定義
BL_STATUS  db     Ø                       敵の弾の状態
BL_XZAHYOU db     Ø                       敵の弾のX座標
BL_YZAHYOU db     Ø                       敵の弾のY座標
BL_HOUKOU  db     Ø                       敵の弾の方向
bl_info  ends                             構造体定義終了
;
EMBVAL  equ     4Ø                        EneMy Bullet VALue
EMBWOK  db      EMBVAL * type bl_info dup (Ø)   EneMy Bullet Work area
;
mb_info struc   ;My Bullet INFOrmation    ── 敵の弾情報の構造体定義
MB_STATUS  db     Ø                       自機の弾の状態
MB_XZAHYOU db     Ø                       自機の弾のX座標
MB_YZAHYOU db     Ø                       自機の弾のY座標
mb_info  ends                             構造体定義終了
;
MYBVAL  equ     12                        MY Bullet VALue
MYBWOK  db      MYBVAL * type mb_info dup (Ø)   MY Bullet Work area
;
REND    equ     8Ø-4                      Right END
DEND    equ     178                       Down END
```

```
;
MYLOC   dw      Ø                                   MY LOCation
MYRST   db      Ø                                   MY ReST
SSKEY   db      Ø                                   Set Space KEY
;
STOP    equ     Ø                                   STOP
QRR     equ     1                                   Direction=1
QUR     equ     2                                   Direction=2
QUU     equ     3                                   Direction=3
QUL     equ     4                                   Direction=4
QLL     equ     5                                   Direction=5
QDL     equ     6                                   Direction=6
QDD     equ     7                                   Direction=7
QDR     equ     8                                   Direction=8
CHIJOU  equ     9                                   Direction=9
;
S1      equ     1ØH                                 Shot 1
S2      equ     2ØH                                 Shot 2
S3      equ     3ØH                                 Shot 3
;
_END    equ     8ØH                                 END     command of QRL
_JUMP   equ     81H                                 JUMP    command of QRL
_IFZ    equ     82H                                 IFZ     command of QRL
_IFC    equ     83H                                 IFC     command of QRL
_CALL   equ     84H                                 CALL    command of QRL
_FETCH  equ     85H                                 FETCH   command of QRL
;
DSCOR   db      1BH,"[1;1H",1BH,"[21;33mＳＫＹ　ＢＲＵＩＳＥＲ"
MSCOR   db      1BH,"[1;5ØH",1BH,"[23;37m   得点  Ø          "
        db      1BH,"[1;72H",1BH,"[2Ø;32m残り "
MYREM   db      1BH,"[1;77H"
MYREN   db      "8 機",1BH,"[23;37m$"
;
lodinf  struc                                       ロード情報テーブル構造体定義
CMD     db      Ø
PASS    db      Ø
ENDSIN  db      Ø
LDADR   dw      Ø
LDSEG   dw      Ø
lodinf  ends
;
LODTBL: ;LOaD TaBLe
        lodinf <1,"MOJI.DAT",Ø,Ø,MOJSEG>
        lodinf <1,"SKYBRU.DAT",Ø,Ø,PTNSEG>
        lodinf <1,"MAPPAT.DAT",Ø,Ø,MPPSEG>
        lodinf <1,"MAPDAT2.DAT",Ø,Ø,MPDSEG>
        lodinf < >
;
        include LIST6-3.ASM
```

● Appendix

A.1 ツールの入力

このツールはダンプ・リストの形態で載せてありますので，MS-DOS用のシンボリックデバッガ「SYMDEB」で打ち込んで使用してください．SYMDEBはMS-DOS，あるいはMASMに添付されています．

また，プログラムは「.EXE」タイプの実行形式となっていますが，SYMDEBのコマンドWで編集データをセーブする時に，ファイル名として拡張子の「.EXE」が使えません．ですから，いったん仮のファイル名でディスクにセーブしておき，ダンプリストをすべて打ち込んだ後，MS-DOSのREN命令で拡張子を「.EXE」に変更してください．図A-1，図A-2にパターン・エディタ，マップ・エディタの各入力例を示します．ただし，パターン・エディタはワークエリアを確保するため，3DE0番地～9FC0番地を0で埋めます．

```
A>SYMDEB ⏎……………SYMDEBを起動する
Microsoft (R) Symbolic Debug Utility  Version 4.ØØ
Copyright (C) Microsoft Corp 1984, 1985.  All rights reserved.

Processor is [8Ø286]

-E DS:ØØØØ ⏎……………コマンドEで空いているメモリ領域を利用してダンプリストを打ち込む
4632:ØØØØ  ØØ.4D   FF.5A   ØØ.CØ   FF.
                 :
                 : ダンプリスト入力
                 :
4632:3DD8  ØØ.5B   FF.3E   ØØ.35   FF.6C   ØØ.1B   FF.29   ØØ.3Ø   FF.24
4632:3DEØ  ØØ.⏎
-F 3DEØ 9FCØ Ø⏎……………コマンドFで3DE0番地～9FC0番地を0で埋める
-N MSPTER.ABC ⏎……………コマンドNで仮のファイル名を設定する
-R CX⏎
CX ØØØØ
:9FCØ⏎
-R BX⏎          } ダンプリストの大きさをBX：CXレジスタに設定する
BX ØØØØ
:Ø ⏎
-W DS:ØØØØ⏎……………コマンドWでダンプリストをディスクに保存
-Q ⏎……………SYMDEBを終了する

A>REN MSPTER.ABC MSPTER.EXE ⏎……………ファイルの拡張子を「.EXE」に変える
A>MSPTER ⏎……………実行
```

図A-1 パターン・エディタの入力

```
A>SYMDEB ↵ ･･････････････SYMDEB を起動する
Microsoft (R) Symbolic Debug Utility  Version 4.ØØ
Copyright (C) Microsoft Corp 1984, 1985.  All rights reserved.

Processor is [8Ø286]

-E DS:ØØØØ ↵ ･･････････････コマンド E で空いているメモリ領域を利用してダンプリストを打ち込む
4632:ØØØØ  ØØ.4D  FF.5A  ØØ.35  FF.
                  :
                  :  ダンプリスト入力
                  :
4632:233Ø  ØØ.C8  FF.ØØ  ØØ.96  FF.18  ØØ.F6 ↵
-N MAPEDIT.ABC ↵ ･･････････････コマンド N で仮のファイル名を設定する
-R CX ↵      ┐
CX ØØØØ      │
:2335 ↵      │
-R BX ↵      ├ ダンプリストの大きさを BX：CX レジスタに設定する
BX ØØØØ      │
:Ø ↵         ┘
-W DS:ØØØØ ↵ ･･････････････コマンド W でダンプリストをディスクに保存
-Q ↵ ･･････････････SYMDEB を終了する

A>REN MAPEDIT.ABC MAPEDIT.EXE ↵ ･･････････････ファイルの拡張子を「EXE」に変える
A>MAPEDIT ↵ ･･････････････実行
```

図 A-2　マップ・エディタの入力

A.2 パターン・エディタ MSPTER

パターン・エディタ「MSPTER」は，各章で使用するキャラクター・パターンをデザインするためのツールです．Appendix 1の手順で，巻末のリストA-1を〝MSPTER.EXE″というファイル名で作成してください．

MSPTERが起動すると，画面上部にはMSPTERが立ち上がったことを示すメッセージが表示され，コマンド待ちの状態になります．

```
[A]▮
```

左側の英字は，ロード／セーブしたりする際のドライブを表しています．このプロンプトが出たら，MSPTERがあなたのコマンド入力を待っているという意味です．まず，HELPを押してください．画面には以下のようなメニューが現れます．

```
A    DRIVE = A
B    DRIVE = B
C    DRIVE = C
D    DRIVE = D
E    Edit
F    DIR *.*
K    Kill file
L    Load file
M    To Ms-dos
S    Save file
```

ここに示された内容と，画面をクリアするスペースキーがコマンドモード時のコマンドのすべてです．各コマンドは対話式になっており，必要に応じあなたに質問をしてきますから，メッセージに合わせて数値あるいは文字を入力してください．一部例外を除き，ESCで質問からの回避ができるようになっています．質問によってはリターンキーを押す必要がないものもありますが，原則的にはリターンキーにより質問に応じたとみなされます．では，各コマンドの説明をします．

● A～D

ロード／セーブなどディスク操作に関するドライブの指定です．

● E(Edit)

エディットモードにはいり，パターンをスクリーンエディットできるようになります．

● F(DIR *.*)

ディスクに保存されている全ファイル名を表示します．

● K(Kill file)

指定されたドライブにあるファイルを削除します．ファイル名を聞いてきますので，間違いのないよう入力してください．

● L(Load file)

指定されたドライブにあるデータファイルをロードします．画面左上にファイル指定用のカーソルが表れますから，矢印キー(↑ ↓ ← →)で選択してください．ファイルの決定は，リターンキーが押された時とします．操作を打ち切りたい場合には，ESC を押してください．また，ファイル名は手操作でも入力できるようになっています，この場合にはワイルド･カードが使用できるようになっています．

● M(To MS-DOS)

『MSPTER.EXE』を終了し，MS-DOS システムへ制御を移します．

● S(Save file)

テキストエリアにあるデータを，指定されたドライブのディスクにセーブします．

■エディットモード

コマンド待ちの状態から，E を押してエディットモードにはいると，画面が切り替わります．

右上には 1～12 までのパターンが表示されます．また，画面左側には編集中のパターンのドットを拡大した状態が表示され，四角い枠のカーソルが点滅します．このカーソルをテンキーの 1 ～ 4 ，6 ～ 9 で移動してスペースキーでドット情報を入力していきます．また，エディットモードの時の各コマンドについては本文 p.49 の表 2-1 を参照してください．

エディットモードから抜けるには ESC を押します．

A.3 マップ・エディタ MAPEDIT

マップ・エディタ「MAPEDIT」は，6章で使う背景を編集するためのツールです．したがって，6章までは使うことはありません．また，このツールでは，背景を編集すると共に，敵をどこに配置するかという情報もセットできるようになっています．

このツールもパターン・エディタと同様に，SYMDEBで巻末のリストA-2を"MAPEDIT.EXE"というファイル名で作成します．

実際にプログラムを実行させる場合には，あらかじめ，マップを構成する基本パターン（地形）を，パターン・エディタで作っておきます．

マップの基本パターンには透明色は必要ありませんから，パターン・エディタのコマンドDでBRGモードを選択してください．サイズは32×32ドット固定ですから，コマンドSでパターンサイズを設定してください．なお，パターン数は，最大96種類まで読み込めるようになっています．

敵の配置を行う場合には，敵のパターンも用意しておいてください．敵のパターンは透明色を必要としますから，コマンドDでT1BRGを選択します．また，敵のサイズも，32×32固定としました．パターン数は同様に，96種類まで読み込めます．

コマンドは全部で7種類あります．CTRL を押すと下に示したようなコマンドテーブルが画面左上に現れますから，テンキーの 2 8 で選択してください．CTRL を離すとコマンドを実行します．

編集
MAP パターン・ロード
MAP パターン・追加
MAP パターン・セーブ
マップ・データ・ロード
マップ・データ・セーブ
敵パターン・ロード
敵パターン・追加
敵パターン・セーブ

追加コマンドは以前にロードしたパターンの後にロードしたい時に使います．なお，セーブする場合には，追加分もまとめて1つのファイルとなります．

各パターンの配置は，テンキーの 2 4 6 8 でカーソルを移動して，リターンキーで決定します．背景と敵パターンの選択は CAPS で使い分けをし，SHIFT ＋テンキーの 2 4 6 8 で各パターンの選択をします．

リスト A-1 MSPTER

```
ØØØØ  4D 5A CØ Ø1 5Ø ØØ Ø4 ØØ-2Ø ØØ FØ Ø7 FF FF DC Ø9
ØØ1Ø  ØØ Ø2 Ø8 C4 ØØ ØØ ØØ ØØ-1E ØØ ØØ ØØ Ø1 ØØ CB Ø7
ØØ2Ø  ØØ ØØ F7 Ø7 ØØ ØØ 73 11-ØØ ØØ 96 11 ØØ ØØ ØØ ØØ
ØØ3Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØ4Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØ5Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØ6Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØ7Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØ8Ø  ØØ ØD ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØ9Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ

ØØAØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØBØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØCØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØDØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØEØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØFØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø1ØØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø11Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø12Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø13Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ

Ø14Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø15Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø16Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø17Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø18Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø19Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø1AØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø1BØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø1CØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø1DØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ

Ø1EØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø1FØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø2ØØ  FC 2E C6 Ø6 9E 17 ØØ 9Ø-2E C6 Ø6 87 17 Ø8 9Ø B8
Ø21Ø  ØØ 4Ø CD 18 B8 ØØ 42 B9-ØØ CØ CD 18 8C C8 8E D8
Ø22Ø  8E CØ Ø6 BA CD ØØ B8 Ø6-35 CD 21 89 1E E2 ØØ 8C
Ø23Ø  Ø6 E4 ØØ Ø7 B8 Ø6 25 CD-21 BØ ØØ A2 A8 12 B4 19
Ø24Ø  CD 21 Ø4 41 A2 C3 12 A2-F4 15 A2 C3 12 A2 DA 12
Ø25Ø  A2 FØ 12 A2 3A 14 A2 89-14 E8 89 21 BA 8F 13 E8
Ø26Ø  6C ØØ BA 9D 13 E8 66 ØØ-AØ C3 12 A2 3A 14 BA 3Ø
Ø27Ø  14 E8 5A ØØ B8 ØØ ØC CD-21 B4 ØØ CD 18 8Ø FC 3F

Ø28Ø  75 Ø5 BØ Ø1 EB Ø8 9Ø 8Ø-FC 3E 75 Ø2 BØ ØC BF 5E
Ø29Ø  Ø1 8B ØE 5C Ø1 F2 AE 75-DB D1 E1 BB 8E Ø1 Ø3 D9
Ø2AØ  FF 17 EB BE 8C C8 8E D8-BA Ø6 17 E8 2Ø ØØ BA EE
Ø2BØ  13 E8 1A ØØ B4 41 CD 18-2E C5 16 E2 ØØ B8 Ø6 25
Ø2CØ  CD 21 B8 ØØ ØC CD 21 BØ-ØØ B4 4C CD 21 CF 57 8B
Ø2DØ  FA 2E 8A 15 8Ø FA 24 74-Ø7 B4 Ø6 CD 21 47 EB F1
Ø2EØ  5F C3 ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ E8 9B Ø6 BA
Ø2FØ  EE Ø1 E8 D9 FF E8 92 Ø6-BA Ø6 Ø2 E8 DØ FF E8 89
Ø3ØØ  Ø6 BA 19 Ø2 E8 C7 FF E8-8Ø Ø6 BA 2C Ø2 E8 BE FF
Ø31Ø  E8 77 Ø6 BA 3F Ø2 E8 B5-FF E8 6E Ø6 BA 52 Ø2 E8

Ø32Ø  AC FF E8 65 Ø6 BA 65 Ø2-E8 A3 FF E8 5C Ø6 BA 78
Ø33Ø  Ø2 E8 9A FF E8 53 Ø6 BA-8B Ø2 E8 91 FF E8 4A Ø6
Ø34Ø  BA 9E Ø2 E8 88 FF E8 41-Ø6 BA B1 Ø2 E8 7F FF E8
Ø35Ø  38 Ø6 BA C4 Ø2 E8 76 FF-E8 2F Ø6 C3 3Ø ØØ 42 62
Ø36Ø  BA BA 41 61 C1 C1 45 65-B2 A8 4D 6D D3 D3 2Ø 2Ø
Ø37Ø  2Ø 2Ø 46 66 CA CA 4C 6C-D8 D8 53 73 C4 C4 43 63
Ø38Ø  BF BF 4B 6B C9 C9 Ø1 Ø1-Ø1 Ø1 44 64 BC BC AF Ø4
Ø39Ø  AF Ø4 AF Ø4 AF Ø4 EC ØØ-EC ØØ EC ØØ EC ØØ E7 Ø2
Ø3AØ  E7 Ø2 E7 Ø2 E7 Ø2 AA Ø4-AA Ø4 AA Ø4 AA Ø4 7Ø ØC
Ø3BØ  7Ø ØC 7Ø ØC 7Ø ØC 4C Ø9-4C Ø9 4C Ø9 4C Ø9 32 Ø6

Ø3CØ  32 Ø6 32 Ø6 32 Ø6 2F Ø5-2F Ø5 2F Ø5 2F Ø5 A4 ØØ
Ø3DØ  A4 ØØ A4 ØØ A4 ØØ 95 21-95 21 95 21 95 21 AØ Ø4
Ø3EØ  AØ Ø4 AØ Ø4 AØ Ø4 A5 Ø4-A5 Ø4 A5 Ø4 A5 Ø4 1B 5B
Ø3FØ  3E 35 68 2D 2D 2D 2D 2D-2D 2D 2D 2D 2D 2D 2D 2D
Ø4ØØ  2D 2D 2D 2D 2D 24 2Ø 2Ø-41 2Ø A5 A5 A5 2Ø 44 52
Ø41Ø  49 56 45 2Ø 41 2Ø 2Ø 2Ø-24 2Ø 2Ø 42 2Ø A5 A5 A5
Ø42Ø  2Ø 44 52 49 56 45 2Ø 42-2Ø 2Ø 2Ø 24 2Ø 2Ø 43 2Ø
Ø43Ø  A5 A5 A5 2Ø 44 52 49 56-45 2Ø 43 2Ø 2Ø 2Ø 24 2Ø
Ø44Ø  2Ø 44 2Ø A5 A5 A5 2Ø 44-52 49 56 45 2Ø 44 2Ø 2Ø
Ø45Ø  2Ø 24 2Ø 2Ø 45 2Ø A5 A5-A5 2Ø 45 44 49 54 2Ø 2Ø

Ø46Ø  2Ø 2Ø 2Ø 2Ø 24 2Ø 2Ø 46-2Ø A5 A5 A5 2Ø 46 49 4C
Ø47Ø  45 53 2Ø 2A 2E 2A 2Ø 24-2Ø 2Ø 4B 2Ø A5 A5 A5 2Ø
Ø48Ø  4B 49 4C 4C 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 24 2Ø 2Ø 4C 2Ø A5

Ø49Ø  A5 A5 2Ø 4C 4F 41 44 2Ø-46 49 4C 45 2Ø 24 2Ø 2Ø
Ø4AØ  4D 2Ø A5 A5 A5 2Ø 54 4F-2Ø 4D 53 2D 44 4F 53 2Ø
Ø4BØ  24 2Ø 2Ø 53 2Ø A5 A5 A5-2Ø 53 41 56 45 2Ø 46 49
Ø4CØ  4C 45 2Ø 24 2D 2D 2D 2D-2D 2D 2D 2D 2D 2D 2D 2D
Ø4DØ  2D 2D 2D 2D 2D 2D 24 B9-ØØ ØØ F7 E8 F7 E8 F7 E8
Ø4EØ  F7 E8 F7 E8 E2 F4 C3 B9-11 ØØ BE Ø5 13 BF 18 13
Ø4FØ  F3 A4 B9 11 ØØ BF Ø5 13-BØ 2Ø F3 AA E8 ØC ØØ B9

Ø5ØØ  11 ØØ BE 18 13 BF Ø5 13-F3 A4 C3 E8 54 ØF 75 Ø3
Ø51Ø  E9 94 Ø3 BA F6 16 E8 B5-FD E8 6E Ø4 8A ØE A8 12
Ø52Ø  B5 ØØ B8 Ø3 ØØ Ø4 Ø1 37-E2 FB ØD 3Ø 3Ø 86 C4 A3
Ø53Ø  FC 14 A3 4Ø 15 A3 4D 15-BA F5 14 E8 9Ø FD BF Ø5
Ø54Ø  13 C7 Ø6 3C 15 ØØ ØØ C7-Ø6 3A 15 ØØ ØØ C7 Ø6 43
Ø55Ø  15 32 32 C6 Ø6 2B 13 Ø1-9Ø B8 ØØ ØC CD 21 B4 ØØ
Ø56Ø  CD 18 3C ØD 74 1F 3C 1B-75 Ø3 E9 ØF Ø1 8Ø FC 3C
Ø57Ø  75 Ø6 B8 ØØ 38 EB Ø9 9Ø-8Ø FC 3B 75 Ø3 B8 Ø8 ØØ
Ø58Ø  E8 3F Ø7 EB D4 BF Ø5 13-8A 26 C3 12 B9 1Ø ØØ 8A
Ø59Ø  Ø5 3C 3A 74 13 3C 2Ø 74-Ø6 3C 5C 74 Ø2 8A EØ 47

Ø5AØ  E2 ED BE Ø5 13 EB 5Ø 9Ø-8B F7 46 8Ø FC 41 7D Ø3
Ø5BØ  E9 C9 ØØ 8Ø FC 44 7E 13-8Ø FC 61 7D Ø3 E9 BC ØØ
Ø5CØ  8Ø FC 64 7E Ø3 E9 B4 ØØ-8Ø EC 2Ø 88 26 EC Ø4 5Ø
Ø5DØ  BA EC Ø4 B4 3B CD 21 58-72 1D 88 26 C3 12 88 26
Ø5EØ  DA 12 88 26 FØ 12 88 26-3A 14 88 26 89 14 8Ø EC
Ø5FØ  41 8A D4 B4 ØE CD 21 BF-DC 12 B9 ØD ØØ 8A Ø4 3C
Ø6ØØ  ØØ 74 12 3C 2Ø 74 ØE 3C-5C 74 Ø6 3C 3A 74 Ø2 88
Ø61Ø  Ø5 47 46 E2 E8 C6 Ø5 2Ø-C6 Ø5 2Ø C6 Ø5 ØØ BA 33
Ø62Ø  17 B4 1A CD 21 BF DA 12-AØ C3 12 88 Ø5 8B D7 B9
Ø63Ø  ØØ ØØ B4 4E CD 21 72 5D-E8 4F Ø3 BA F7 15 E8 8D

Ø64Ø  FC B8 Ø7 ØC CD 21 3C 59-74 14 3C 79 74 1Ø 3C DD
Ø65Ø  74 ØC E8 35 Ø3 BA 96 15-E8 73 FC EB 1F 9Ø BA DA
Ø66Ø  12 B4 41 CD 21 72 ØC E8-2Ø Ø3 BA 89 15 E8 5E FC
Ø67Ø  EB ØA 9Ø E8 14 Ø3 BA 26-16 E8 52 FC C6 Ø6 2B 13
Ø68Ø  ØØ 9Ø B8 3Ø 33 A3 4Ø 15-A3 4D 15 BA F6 16 E8 3D
Ø69Ø  FC E8 F6 Ø2 C3 E8 F2 Ø2-BA 53 16 E8 3Ø FC EB DC
Ø6AØ  B2 41 EB 1Ø 9Ø B2 42 EB-ØB 9Ø B2 43 EB Ø6 9Ø B2
Ø6BØ  44 EB Ø1 9Ø 88 16 EC Ø4-52 BA EC Ø4 B4 3B CD 21
Ø6CØ  5A 72 25 88 16 F4 15 88-16 C3 12 88 16 DA 12 88
Ø6DØ  16 FØ 12 88 16 3A 14 88-16 89 14 8Ø EA 41 B4 ØE

Ø6EØ  CD 21 BA EF 15 E8 E6 FB-E8 9F Ø2 C3 42 3A 5C ØØ
Ø6FØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø7ØØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø71Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø72Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ BA
Ø73Ø  FB 13 E8 99 FB BØ ØØ A2-A8 12 C3 52 51 53 5Ø B2
Ø74Ø  ØØ B4 36 CD 21 F7 E1 F7-E3 52 51 5Ø 8B D2 8B CØ
Ø75Ø  B9 1Ø 27 F7 F1 51 5Ø 8B-CØ B1 64 F6 F1 51 5Ø 8A
Ø76Ø  CØ 32 E4 B1 64 F6 F1 8A-C4 32 E4 B1 ØA F6 F1 Ø4
Ø77Ø  3Ø A2 ØB Ø6 8Ø C4 3Ø 88-26 ØC Ø6 58 59 51 5Ø 8A

Ø78Ø  C4 32 E4 B1 64 F6 F1 8A-C4 32 E4 B1 ØA F6 F1 Ø4
Ø79Ø  3Ø A2 ØD Ø6 8Ø C4 3Ø 88-26 ØE Ø6 58 59 58 59 51
Ø7AØ  5Ø 8B C2 B1 64 F6 F1 51-5Ø 8A CØ 32 E4 B1 64 F6
Ø7BØ  F1 8A C4 32 E4 B1 ØA F6-F1 Ø4 3Ø A2 ØF Ø6 8Ø C4
Ø7CØ  3Ø 88 26 1Ø Ø6 58 59 51-5Ø 8A C4 32 E4 B1 64 F6
Ø7DØ  F1 8A C4 32 E4 B1 ØA F6-F1 Ø4 3Ø A2 11 Ø6 8Ø C4
Ø7EØ  3Ø 88 26 12 Ø6 58 59 58-59 58 59 5A E8 9B Ø1 BB
Ø7FØ  ØB Ø6 8Ø 3F 3Ø 75 Ø6 C6-Ø7 2Ø 43 EB F5 BA ØB Ø6
Ø8ØØ  E8 CB FA E8 84 Ø1 58 5B-59 5A C3 ØØ ØØ ØØ ØØ ØØ
Ø81Ø  ØØ ØØ ØØ 81 4Ø 83 6F 83-43 83 67 81 4Ø 83 74 83

Ø82Ø  8A 81 5B 24 ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ 81 4Ø 2Ø 2Ø
Ø83Ø  2Ø 24 E8 2D ØC 75 Ø3 EB-6E 9Ø BA 4B 14 E8 8E FA
Ø84Ø  B8 ØØ ØØ A3 A2 12 BA 33-17 B4 1A CD 21 BF FØ 12
Ø85Ø  AØ C3 12 88 Ø5 8B D7 B9-ØØ ØØ B4 4E CD 21 73 Ø3
Ø86Ø  EB 42 9Ø 3D ØC ØØ 75 Ø3-EB 3A 9Ø E8 1C Ø1 E8 3A
Ø87Ø  ØØ BF 33 17 83 C7 1E B8-2Ø ØØ B9 Ø6 ØØ F3 AB B4
Ø88Ø  4F CD 21 72 1F 3D ØC ØØ-74 1A A1 A2 12 4Ø A3 A2
Ø89Ø  12 3D Ø2 ØØ 7C Ø9 B8 ØØ-ØØ A3 A2 12 E8 EB ØØ E8
Ø8AØ  Ø9 ØØ EB CD E8 94 FE E8-EØ ØØ C3 51 52 56 BE 33
Ø8BØ  17 83 C6 1E B9 ØC ØØ 8A-14 B4 Ø6 CD 21 46 E2 F7

Ø8CØ  B2 2Ø B4 Ø6 CD 21 BE 33-17 8B 44 1A 8B 54 1C 52
Ø8DØ  51 5Ø 8B D2 8B CØ B9 1Ø-27 F7 F1 51 5Ø 8B CØ B1
Ø8EØ  64 F6 F1 51 5Ø 8A CØ 32-E4 B1 64 F6 F1 8A C4 32
Ø8FØ  E4 B1 ØA F6 F1 Ø4 3Ø A2-24 Ø6 8Ø C4 3Ø 88 26 25
Ø9ØØ  Ø6 58 59 51 5Ø 8A C4 32-E4 B1 64 F6 F1 8A C4 32
Ø91Ø  E4 B1 ØA F6 F1 Ø4 3Ø A2-26 Ø6 8Ø C4 3Ø 88 26 27
```

```
Ø92Ø  Ø6 58 59 58 59 51 5Ø 8B-C2 B1 64 F6 F1 51 5Ø 8A
Ø93Ø  CØ 32 E4 B1 64 F6 F1 8A-C4 32 E4 B1 ØA F6 F1 Ø4
Ø94Ø  3Ø A2 28 Ø6 8Ø C4 3Ø 88-26 29 Ø6 58 59 51 5Ø 8A
Ø95Ø  C4 32 E4 B1 64 F6 F1 8A-C4 32 E4 B1 ØA F6 F1 Ø4

Ø96Ø  3Ø A2 2A Ø6 8Ø C4 3Ø 88-26 2B Ø6 58 59 58 59 58
Ø97Ø  59 5A BE 24 Ø6 8Ø 3C 3Ø-75 Ø6 C6 Ø4 2Ø 46 EB F5
Ø98Ø  BA 24 Ø6 E8 48 F9 5E 5A-59 C3 AØ A8 12 FE CØ 3A
Ø99Ø  Ø6 2F 14 7C Ø9 BA ØB 14-E8 33 F9 EB ØA 9Ø A2 A8
Ø9AØ  12 BA 9A 13 E8 27 F9 C3-A1 AB 12 4Ø 3B Ø6 BB 12
Ø9BØ  7E ØD C7 Ø6 B1 12 Ø1 ØØ-B8 ØØ ØØ A3 AB 12 C3 A3
Ø9CØ  AB 12 BA 9A 13 E8 Ø6 F9-C3 Ø6 B8 FC Ø9 8E CØ BE
Ø9DØ  33 17 83 C6 1E 2E 8B 3E-AD 12 2E 89 3E AF 12 B9
Ø9EØ  Ø8 ØØ F3 A5 2E 3B 3E B3-12 7C Ø3 BF ØØ ØØ 2E 89
Ø9FØ  3E AD 12 Ø7 C3 Ø6 B8 FC-Ø9 8E CØ 33 CØ 8B F8 A3

ØAØØ  B1 12 B9 ØØ 2Ø F3 AB Ø7-BA 6A 14 E8 CØ F8 B8 ØØ
ØA1Ø  ØØ A3 A2 12 A3 AD 12 A3-A9 12 A3 AB 12 A3 B9 12
ØA2Ø  B4 31 88 26 Ø2 17 BØ 3Ø-A2 Ø1 17 A2 FE 16 BØ 34
ØA3Ø  A2 FF 16 BA 33 17 B4 1A-CD 21 BF C3 12 AØ C3 12
ØA4Ø  88 Ø5 8B D7 B9 ØØ ØØ B4-4E CD 21 73 Ø3 E9 88 ØØ
ØA5Ø  BA 6A 14 E8 78 F8 E8 4F-FF 51 52 56 BE 33 17 83
ØA6Ø  C6 1E B9 ØC ØØ 8A 14 B4-Ø6 CD 21 46 E2 F7 B2 2Ø
ØA7Ø  B4 Ø6 CD 21 5E 5A 59 E8-4F FF BF 33 17 83 C7 1E
ØA8Ø  B8 2Ø ØØ B9 Ø6 ØØ F3 AB-B4 4F CD 21 73 Ø3 E9 81
ØA9Ø  ØØ A1 A2 12 4Ø A3 A2 12-3B Ø6 BD 12 7C 1Ø B8 ØØ

ØAAØ  ØØ A3 A2 12 83 3E B1 12-ØØ 75 28 E8 FA FE 83 3E
ØABØ  B1 12 ØØ 75 1E 51 52 56-BE 33 17 83 C6 1E B9 ØC
ØACØ  ØØ 8A 14 B4 Ø6 CD 21 46-E2 F7 B2 2Ø B4 Ø6 CD 21
ØADØ  5E 5A 59 E8 F3 FE EB A2-E8 54 FC BA 6F 16 E8 ED
ØAEØ  F7 E8 A6 FE E8 32 ØØ E8-AØ FE BF C5 12 B9 ØC ØØ
ØAFØ  BØ 2Ø F3 AA BF C5 12 C6-Ø5 2A C6 45 Ø1 2E C6 45
ØBØØ  Ø2 2A B9 1Ø ØØ BØ 2Ø BF-Ø5 13 88 Ø5 47 E2 FB EB
ØB1Ø  Ø7 9Ø C7 Ø6 BF 12 Ø1 ØØ-C3 BF C5 12 B9 ØD ØØ B2
ØB2Ø  5B B4 Ø6 CD 21 B2 2Ø B4-Ø6 CD 21 8A 15 8Ø FA 2Ø
ØB3Ø  74 ØC 8Ø FA ØØ 74 Ø7 B4-Ø6 CD 21 47 EØ ED B2 2Ø

ØB4Ø  B4 Ø6 CD 21 B2 5D B4 Ø6-CD 21 C3 ØØ B9 11 ØØ BE
ØB5Ø  Ø5 13 BF 18 13 F3 A4 C6-Ø6 4B Ø9 ØØ E8 13 ØØ 8Ø
ØB6Ø  3E 4B Ø9 Ø1 74 ØB B9 11-ØØ BE 18 13 BF Ø5 13 F3
ØB7Ø  A4 C3 E8 ED Ø8 75 Ø3 E9-2D FD B8 ØØ ØØ A3 BF 12
ØB8Ø  C6 Ø6 4B Ø9 ØØ E8 6D FE-83 3E BF 12 ØØ 75 ØD BØ
ØB9Ø  Ø1 A2 A8 12 C6 Ø6 4B Ø9-ØØ E9 25 Ø1 B8 ØØ ØØ A3
ØBAØ  6Ø 12 A3 A4 12 A3 A6 12-A3 AB 12 A3 A9 12 E8 99
ØBBØ  Ø7 B8 ØØ ØC CD 21 B8 ØØ-ØØ CD 18 BF 84 12 8B ØE
ØBCØ  5E 12 F2 AF 74 Ø5 E8 F9-ØØ EB E6 5Ø BA F6 16 E8
ØBDØ  FC F6 58 8Ø 3E 2B 13 ØØ-75 Ø3 E9 DØ ØØ C6 Ø6 2B

ØBEØ  13 ØØ 9Ø 3C ØD 74 Ø3 E9-C3 ØØ BF Ø5 13 8A 26 C3
ØBFØ  12 B9 1Ø ØØ 8A Ø5 3C 3A-74 13 3C 2Ø 74 Ø6 3C 5C
ØCØØ  74 Ø2 8A EØ 47 E2 ED BF-Ø5 13 EB 67 9Ø 47 8Ø FC
ØC1Ø  41 7C 6Ø 8Ø FC 44 7F 22-88 26 EC Ø4 5Ø BA EC Ø4
ØC2Ø  B4 3B CD 21 58 72 4C 88-26 C3 12 88 26 FØ 12 88
ØC3Ø  26 3A 14 88 26 89 14 EB-31 9Ø 8Ø FC 61 7C 34 8Ø
ØC4Ø  FC 64 7F 2F 8Ø EC 2Ø 88-26 EC Ø4 5Ø BA EC Ø4 B4
ØC5Ø  3B CD 21 58 72 1D 88 26-C3 12 88 26 DA 12 88 26
ØC6Ø  FØ 12 88 26 3A 14 88 26-89 14 8Ø EC 41 8A D4 B4
ØC7Ø  ØE CD 21 AØ C3 12 A2 DA-12 BE DC 12 B9 ØD ØØ 87

ØC8Ø  F7 F3 A4 BØ 2A BF Ø5 13-B9 1Ø ØØ F2 AE 74 ØC BØ
ØC9Ø  3F BF Ø5 13 B9 1Ø ØØ F2-AE 75 ØE BE DC 12 BF C5
ØCAØ  12 B9 ØD ØØ F3 A4 E9 C9-FE E8 21 Ø7 C3 D1 E1 BB
ØCBØ  9Ø 12 Ø3 D9 FF 17 2E 83-3E 6Ø 12 ØØ 75 Ø3 E9 FØ
ØCCØ  FE C3 8Ø 3E 2B 13 ØØ 75-3C 5Ø BØ 2Ø B9 1Ø ØØ BF
ØCDØ  Ø5 13 F3 AA C7 Ø6 3C 15-ØØ ØØ C7 Ø6 3A 15 ØØ ØØ
ØCEØ  C7 Ø6 43 15 32 32 BA 38-15 E8 E2 F5 BA 3E 15 E8
ØCFØ  DC F5 BA 47 15 E8 D6 F5-BA FØ 16 E8 DØ F5 C6 Ø6
ØDØØ  2B 13 Ø1 9Ø 58 3C Ø8 74-64 3D ØØ 38 75 Ø3 E9 89
ØD1Ø  ØØ 3D ØØ 39 75 Ø3 E9 B7-ØØ A8 8Ø 75 Ø7 3C 2Ø 73

ØD2Ø  Ø3 E9 21 Ø1 8B DØ BF Ø5-13 A1 3C 15 Ø3 F8 4Ø 3D
ØD3Ø  1Ø ØØ 7C Ø9 BA 38 15 E8-94 F5 E9 Ø8 Ø1 3B Ø6 3A
ØD4Ø  15 7C Ø3 A3 3A 15 A3 3C-15 88 15 B4 Ø2 CD 21 2E
ØD5Ø  A1 43 15 86 C4 25 ØF ØF-Ø4 Ø1 37 ØD 3Ø 3Ø 86 C4
ØD6Ø  2E A3 43 15 BA 3E 15 E8-64 F5 E9 D8 ØØ A1 3C 15
ØD7Ø  48 3D ØØ ØØ 7D Ø3 E9 CC-ØØ A3 3C 15 2E A1 43 15
ØD8Ø  86 C4 25 ØF ØF 2C Ø1 3F-ØD 3Ø 3Ø 86 C4 2E A3 43
ØD9Ø  15 BA 3E 15 E8 37 F5 E9-AB ØØ A1 3C 15 4Ø 3D 1Ø
ØDAØ  ØØ 7C Ø3 E9 9F ØØ 3B Ø6-3A 15 7E Ø3 E9 96 ØØ A3
ØDBØ  3C 15 2E A1 43 15 86 C4-25 ØF ØF Ø4 Ø1 37 ØD 3Ø

ØDCØ  3Ø 86 C4 2E A3 43 15 BA-3E 15 E8 Ø1 F5 EB 76 9Ø
ØDDØ  BA F6 16 E8 F8 F4 BA 3E-15 E8 F2 F4 BA 47 15 E8
ØDEØ  EC F4 A1 3A 15 48 3D ØØ-ØØ 7C 5A 3B Ø6 3C 15 7C
```

```
ØDFØ  Ø6 A3 3A 15 EB 1C 9Ø A3-3A 15 A3 3C 15 2E A1 43
ØEØØ  15 86 C4 25 ØF ØF 2C Ø1-3F ØD 3Ø 3Ø 86 C4 2E A3
ØE1Ø  43 15 B9 1Ø ØØ BF Ø5 13-A1 3C 15 Ø3 F8 2B C8 8A
ØE2Ø  45 Ø1 88 Ø5 47 E2 F8 C6-Ø5 2Ø BA 4B 15 E8 9E F4
ØE3Ø  BF Ø5 13 B9 1Ø ØØ 8A 15-B4 Ø2 CD 21 47 E2 F7 BA
ØE4Ø  3E 15 E8 89 F4 BA FØ 16-E8 83 F4 C3 95 D2 8F 57
ØE5Ø  83 66 81 5B 83 5E 82 CD-2Ø 24 ØØ ØØ ØØ ØØ ØØ ØØ

ØE6Ø  ØØ ØØ 2Ø 83 6F 83 43 83-67 82 C5 82 B7 81 42 24
ØE7Ø  B9 11 ØØ BE Ø5 13 BF 18-13 F3 A4 E8 ØC ØØ B9 11
ØE8Ø  ØØ BE 18 13 BF Ø5 13 F3-A4 C3 E8 D5 Ø5 75 Ø3 E9
ØE9Ø  15 FA AØ 82 17 B4 ØØ 8A-1E 83 17 8A FC BA ØØ ØØ
ØEAØ  F7 E3 8A 1E 84 17 Ø2 1E-9E 17 BA ØØ ØØ F7 E3 BB
ØEBØ  ØC ØØ BA ØØ ØØ F7 E3 A3-9E 12 BA F6 16 E8 ØE F4
ØECØ  A1 9E 12 33 D2 52 51 5Ø-8B D2 8B CØ B9 1Ø 27 F7
ØEDØ  F1 51 5Ø 8B CØ B1 64 F6-F1 51 5Ø 8A CØ 32 E4 B1
ØEEØ  64 F6 F1 8A C4 32 E4 B1-ØA F6 F1 Ø4 3Ø A2 5A ØC
ØEFØ  8Ø C4 3Ø 88 26 5B ØC 58-59 51 5Ø 8A C4 32 E4 B1

ØFØØ  64 F6 F1 8A C4 32 E4 B1-ØA F6 F1 Ø4 3Ø A2 5C ØC
ØF1Ø  8Ø C4 3Ø 88 26 5D ØC 58-59 58 59 51 5Ø 8B C2 B1
ØF2Ø  64 F6 F1 51 5Ø 8A CØ 32-E4 B1 64 F6 F1 8A C4 32
ØF3Ø  E4 B1 ØA F6 F1 Ø4 3Ø A2-5E ØC 8Ø C4 3Ø 88 26 5F
ØF4Ø  ØC 58 59 51 5Ø 8A C4 32-E4 B1 64 F6 F1 8A C4 32
ØF5Ø  E4 B1 ØA F6 F1 Ø4 3Ø A2-6Ø ØC 8Ø C4 3Ø 88 26 61
ØF6Ø  ØC 58 59 58 59 58 59 5A-BA 4C ØC E8 6Ø F3 BØ 3Ø
ØF7Ø  BF 5A ØC B9 Ø8 ØØ FC F3-AE 4F 8B D7 B4 Ø9 CD 21
ØF8Ø  E8 Ø7 FA 8A ØE A8 12 B5-ØØ B8 Ø3 ØØ Ø4 Ø1 37 E2
ØF9Ø  FB ØD 3Ø 3Ø 86 C4 A3 B9-14 A3 4Ø 15 A3 4D 15 BA

ØFAØ  B2 14 E8 29 F3 BF Ø5 13-C7 Ø6 3C 15 ØØ ØØ C7 Ø6
ØFBØ  3A 15 ØØ ØØ C7 Ø6 43 15-32 32 B9 1Ø ØØ 8A 15 8Ø
ØFCØ  FA 2Ø 74 29 8Ø FA ØØ 74-24 B4 Ø2 CD 21 FF Ø6 3A
ØFDØ  15 FF Ø6 3C 15 2E A1 43-15 86 C4 25 ØF ØF Ø4 Ø1
ØFEØ  37 ØD 3Ø 3Ø 86 C4 2E A3-43 15 47 E2 DØ C6 Ø6 2B
ØFFØ  13 Ø1 9Ø B8 ØØ ØC CD 21-B4 ØØ CD 18 3C ØD 74 1F
1ØØØ  3C 1B 75 Ø3 E9 41 Ø1 8Ø-FC 3C 75 Ø6 B8 ØØ 38 EB
1Ø1Ø  Ø9 9Ø 8Ø FC 3B 75 Ø3 B8-Ø8 ØØ E8 A5 FC EB D4 BF
1Ø2Ø  Ø5 13 8A 26 C3 12 B9 1Ø-ØØ 8A Ø5 3C 3A 74 13 3C
1Ø3Ø  2Ø 74 Ø6 3C 5C 74 Ø2 8A-EØ 47 E2 ED BE Ø5 13 EB

1Ø4Ø  5Ø 9Ø 8B F7 46 8Ø FC 41-7D Ø3 E9 FB ØØ 8Ø FC 44
1Ø5Ø  7E 13 8Ø FC 61 7D Ø3 E9-EE ØØ 8Ø FC 64 7E Ø3 E9
1Ø6Ø  E6 ØØ 8Ø EC 2Ø 88 26 EC-Ø4 5Ø BA EC Ø4 B4 3B CD
1Ø7Ø  21 58 72 1D 88 26 C3 12-88 26 DA 12 88 26 FØ 12
1Ø8Ø  88 26 3A 14 88 26 89 14-8Ø EC 41 8A D4 B4 ØE CD
1Ø9Ø  21 BF DC 12 B9 ØD ØØ F3-A4 BA 33 17 B4 1A CD 21
1ØAØ  BF DA 12 AØ C3 12 88 Ø5-8B D7 B9 ØØ ØØ B4 4E CD
1ØBØ  21 72 1D E8 D4 F8 BA 6Ø-15 E8 12 F2 B8 Ø7 ØC CD
1ØCØ  21 3C 59 74 ØB 3C 79 74-Ø7 3C DD 74 Ø3 EB 79 9Ø
1ØDØ  B2 ØØ B4 36 CD 21 F7 E1-F7 E3 23 D2 75 12 2B Ø6

1ØEØ  9E 12 73 ØC E8 A3 F8 BA-D3 15 E8 E1 F1 EB 59 9Ø
1ØFØ  B4 3C BA DA 12 B9 ØØ ØØ-CD 21 72 43 A3 AØ 12 8B
11ØØ  D8 BA EØ 3B 8B ØE 9E 12-B4 4Ø CD 21 9C B4 3E 8B
111Ø  1E AØ 12 CD 21 9D 72 27-4Ø 48 75 ØC E8 6B F8 BA
112Ø  D3 15 E8 A9 F1 EB 21 9Ø-E8 5F F8 BA 89 15 E8 9D
113Ø  F1 B9 11 ØØ BE Ø5 13 BF-18 13 F3 A4 EB ØA 9Ø E8
114Ø  48 F8 BA BE 15 E8 86 F1-C6 Ø6 2B 13 ØØ 9Ø B8 3Ø
115Ø  33 A3 4Ø 15 A3 4D 15 BA-F6 16 E8 71 F1 E8 2A F8
116Ø  C3 A1 A9 12 4Ø 3B Ø6 BD-12 7C Ø6 E8 DC Ø1 EB 33
117Ø  9Ø 2E 8B 1E A4 12 83 C3-1Ø 2E 3B 1E AD 12 7D 23

118Ø  A3 A9 12 BA FC 16 E8 45-F1 E8 DD Ø1 2E 3B 1E B5
119Ø  12 7C Ø5 2E 2B 1E B5 12-2E 89 1E A4 12 E8 Ø4 ØØ
11AØ  E8 A7 Ø1 C3 2E A1 Ø1 17-25 ØF ØF 86 C4 Ø4 Ø3 37
11BØ  ØD 3Ø 3Ø FE C4 86 C4 2E-A3 Ø1 17 C3 A1 A9 12 48
11CØ  3D ØØ ØØ 7D Ø6 E8 82 Ø1-EB 2F 9Ø 2E 8B 1E A4 12
11DØ  83 EB 1Ø 83 FB ØØ 7C 21-A3 A9 12 BA FC 16 E8 ED
11EØ  FØ E8 85 Ø1 83 FB ØØ 7D-Ø5 2E Ø3 1E B5 12 2E 89
11FØ  1E A4 12 E8 Ø4 ØØ E8 51-Ø1 C3 2E A1 Ø1 17 25 ØF
12ØØ  ØF 86 C4 2C Ø3 3F ØD 3Ø-3Ø FE CC 86 C4 2E A3 Ø1
121Ø  17 C3 A1 AB 12 48 3D ØØ-ØØ 7D 2D 2E A1 A4 12 2D

122Ø  4Ø ØØ 3D ØØ ØØ 7D Ø3 EB-5F 9Ø 5Ø E8 3B Ø1 BA DA
123Ø  16 E8 9A FØ 58 3D ØØ ØØ-7D Ø5 2E Ø3 Ø6 B5 12 E8
124Ø  47 ØØ E8 Ø5 Ø1 EB 41 9Ø-2E 8B 1E A4 12 83 EB 4Ø
125Ø  83 FB ØØ 7D Ø3 EB 31 9Ø-A3 AB 12 BA FC 16 E8 6D
126Ø  FØ E8 Ø5 Ø1 83 FB ØØ 7D-Ø5 2E Ø3 1E B5 12 2E 89
127Ø  1E A4 12 2E A1 FE 16 86-C4 2C 31 3F ØD 3Ø 3Ø 86
128Ø  C4 2E A3 FE 16 E8 C2 ØØ-C3 5Ø 2E FF 36 Ø1 17 8B
129Ø  D8 2E 8B ØE A9 12 41 49-74 Ø8 83 EB 1Ø E8 5A FF
12AØ  E2 F8 B9 ØØ ØØ 2E 89 1E-A4 12 E8 BC ØØ E8 F4 FE
12BØ  83 C3 1Ø 2E 3B 1E AD 12-7D Ø8 41 2E 3B ØE BD 12
```

```
12CØ  7C E3 2E 8F Ø6 Ø1 17 58-2E A3 A4 12 C3 A1 AB 12
12DØ  4Ø 3B Ø6 BB 12 7C 31 2E-A1 A4 12 Ø5 4Ø ØØ 2E 3B
12EØ  Ø6 AD 12 7C Ø3 EB 62 9Ø-5Ø E8 7D ØØ BA C8 16 E8
12FØ  DC EF 58 2E 3B Ø6 B5 12-7C Ø5 2E 2B Ø6 B5 12 E8
13ØØ  87 FF E8 45 ØØ EB 42 9Ø-2E 8B 1E A4 12 83 C3 4Ø
131Ø  2E 3B 1E AD 12 7D 32 A3-AB 12 BA FC 16 E8 AE EF
132Ø  E8 46 ØØ 2E 3B 1E B5 12-7C Ø5 2E 2B 1E B5 12 2E
133Ø  89 1E A4 12 2E A1 FE 16-86 C4 Ø4 31 37 ØD 3Ø 3Ø
134Ø  86 C4 2E A3 FE 16 E8 Ø1-ØØ C3 BA 21 17 E8 7E EF
135Ø  BA FC 16 E8 78 EF E8 39-ØØ BA 7A 14 E8 6F EF E8

136Ø  3Ø ØØ BA 2A 17 E8 66 EF-C3 BA FC 16 E8 5F EF 53
137Ø  51 1E B8 FC Ø9 8E D8 2E-8B 1E A4 12 B9 ØC ØØ 8A
138Ø  17 B4 Ø2 CD 21 43 E2 F7-1F 59 5B BA 2A 17 E8 3D
139Ø  EF C3 53 51 1E B8 FC Ø9-8E D8 2E 8B 1E A4 12 BE
13AØ  Ø5 13 B9 ØC ØØ 8A 17 2E-88 14 B4 Ø2 CD 21 43 46
13BØ  E2 F3 2E C6 Ø4 ØØ 1F 59-5B C3 AØ C3 12 A2 DA 12
13CØ  BE DC 12 BF Ø5 13 B9 ØD-ØØ 87 F7 F3 A4 BA DA 12
13DØ  BØ ØØ B4 3D CD 21 72 6Ø-A3 AØ 12 8B D8 B8 Ø2 42
13EØ  B9 ØØ ØØ 8B D1 CD 21 89-16 5C 12 A3 5A 12 B4 3E
13FØ  8B 1E AØ 12 CD 21 BA DA-12 BØ ØØ B4 3D CD 21 A3

14ØØ  AØ 12 BA EØ 3B 8B ØE 9C-12 8B 1E AØ 12 B4 3F CD
141Ø  21 9C B4 3E 8B 1E AØ 12-CD 21 9D 72 1B BA 6A 14
142Ø  E8 AB EE BA 55 15 E8 A5-EE B8 Ø1 ØØ A3 6Ø 12 BØ
143Ø  Ø1 A2 A8 12 A2 4B Ø9 C3-BA 6A 14 E8 9Ø EE B8 Ø1
144Ø  ØØ A3 6Ø 12 BØ ØØ A2 A8-12 A2 4B Ø9 B9 1Ø ØØ BØ
145Ø  2Ø BF Ø5 13 88 Ø5 47 E2-FB C3 ØØ ØØ ØØ ØØ Ø6 ØØ
146Ø  ØØ ØØ B4 Ø4 AØ C3 12 3C-41 74 Ø4 3C 42 75 14 24
147Ø  ØF Ø4 8F CD 1B 8Ø E4 FØ-8Ø FC 6Ø 75 Ø6 BA 97 16
148Ø  E8 4B EE C3 ØD 1C 1B ØØ-ØØ 3C ØØ 3B ØØ 3A ØØ 3D
149Ø  CD 1Ø 12 1Ø BC ØF 61 ØF-38 12 BA 11 8Ø 43 ØØ ØØ

14AØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
14BØ  ØØ ØØ ØØ ØØ 7D ØØ 7D ØØ-ØØ ØØ ØØ ØA ØØ Ø4 ØØ ØØ
14CØ  ØØ ØØ ØØ 42 3A 2A 2E 44-41 54 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
14DØ  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ ØØ-ØØ ØØ 42 3A 2Ø 2Ø 2Ø 2Ø
14EØ  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø ØØ ØØ ØØ ØØ
14FØ  42 3A 2A 2E 2A 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
15ØØ  2Ø ØØ ØØ ØØ ØØ 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
151Ø  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ ØØ-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
152Ø  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø ØØ ØØ ØØ 2Ø 2Ø 2Ø 2Ø
153Ø  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø

154Ø  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
155Ø  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
156Ø  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø ØD ØA 82 57
157Ø  9Ø 46 92 86 82 57 9Ø 46-83 82 81 5B 83 68 82 FØ
158Ø  91 49 91 FØ 82 B5 82 DC-82 B7 81 42 ØA ØD 24 1B
159Ø  5B 3E 31 68 1B 5B 32 4A-ØD ØA ØD ØA 24 1B 5B 73
15AØ  1B 5B 31 3B 31 48 1B 5B-32 3Ø 3B 33 32 6D 1B 5B
15BØ  3E 35 68 ØØ 3D 3D 2Ø 4D-53 5Ø 54 45 52 2Ø 3D 3D
15CØ  2Ø 28 43 29 31 39 39 3Ø-2Ø 62 79 2Ø 54 2E 48 69
15DØ  64 61 6B 61 2Ø 26 2Ø 4D-2E 41 6F 79 61 6D 61 2Ø

15EØ  76 65 72 2E 31 2E 33 1B-5B 4B 1B 5B 75 24 1B 5B
15FØ  3E 35 6C 1B 5B 3E 31 6C-ØD ØA 24 1B 5B 3E 35 68
16ØØ  1B 5B 33 3B 31 48 1B 5B-3Ø 4A 24 1B 5B 33 3B 31
161Ø  48 1B 5B 31 4D 1B 5B 32-34 3B 31 48 24 1B 5B 33
162Ø  3B 31 48 1B 5B 31 4D 1B-5B 32 34 3B 31 48 24 16
163Ø  Ø7 1B 5B 32 32 3B 33 36-6D 5B 42 5D 2Ø 1B 5B 32
164Ø  33 3B 33 37 6D 1B 5B 3E-35 6C 24 1B 5B 32 31 3B
165Ø  33 33 6D 46 49 4C 45 53-2Ø 2A 2E 2A 1B 5B 32 33
166Ø  3B 33 37 6D 1B 5B 3E 35-68 24 1B 5B 33 3B 31 48
167Ø  1B 5B 3Ø 4A 1B 5B 3E 35-68 24 1B 5B 33 3B 31 48

168Ø  1B 5B 32 31 3B 33 33 6D-5B 42 5D 1B 5B 32 32 3B
169Ø  33 36 6D 2Ø 4C 4F 41 44-2Ø 46 49 4C 45 2Ø 4E 41
16AØ  4D 45 2Ø 3D 2Ø 1B 5B 4B-1B 5B 32 33 3B 33 37 6D
16BØ  Ø7 24 1B 5B 3E 35 68 1B-5B 3Ø 33 3B 31 48 1B 5B
16CØ  32 31 3B 33 33 6D 2Ø 2Ø-2A 1B 5B 32 32 3B 33 36
16DØ  6D 2Ø 53 41 56 45 2Ø 46-49 4C 45 2Ø 4E 41 4D 45
16EØ  2Ø 3D 2Ø 1B 5B 4B 1B 5B-32 33 3B 33 37 6D Ø7 1B
16FØ  5B 3E 35 6C 24 1B 5B 3E-35 68 1B 5B 3Ø 33 3B 31
17ØØ  48 1B 5B 32 31 3B 33 33-6D 2Ø 2Ø 2A 1B 5B 32 32
171Ø  3B 33 36 6D 2Ø 4B 49 4C-4C 2Ø 46 49 4C 45 2Ø 4E

172Ø  41 4D 45 2Ø 3D 2Ø 1B 5B-4B 1B 5B 32 33 3B 33 37
173Ø  6D Ø7 1B 5B 3E 35 6C 24-Ø7 24 ØØ ØØ ØØ ØØ 1B 5B
174Ø  3Ø 33 3B 32 32 48 24 1B-5B 4B 24 1B 5B 3Ø 33 3B
175Ø  32 32 48 24 24 4C 4F 41-44 2Ø 45 4E 44 ØA ØD 24
176Ø  2Ø 2Ø 2Ø 2Ø 93 AF 82 B6-CC A7 B2 D9 82 AA 97 4C
177Ø  82 E8 82 DC 82 B7 A1 8C-7Ø 91 B1 82 B5 82 DC 82
178Ø  B7 82 A9 2Ø 59 2D 4E 3F-24 2Ø 2Ø 2Ø 2Ø 8E CØ 8D
179Ø  73 8F 49 97 B9 24 1B 5B-32 31 3B 33 33 6D 2Ø 2Ø

17AØ  2Ø 2Ø 83 4C 83 83 83 93-83 5A 83 8B 81 4Ø 21 21
17BØ  1B 5B 32 33 3B 33 37 6D-1B 5B 3E 35 68 24 2Ø 2Ø

17CØ  2Ø 2Ø 83 5A 81 5B 83 75-81 4Ø 83 47 83 89 81 5B
17DØ  2Ø 21 21 2Ø 2Ø 2Ø 2Ø 83-66 83 42 83 58 83 4E 82
17EØ  AA A4 88 EA 94 74 82 C5-82 B7 81 42 21 21 24 1B
17FØ  5B 3E 35 68 41 3A 24 1B-5B 32 31 3B 33 33 6D 2Ø
18ØØ  2Ø 2Ø 2Ø 83 74 83 4Ø 83-43 83 8B 8A 6D 94 46 2Ø
181Ø  4F 4B 2Ø 59 2D 4E 2Ø 3F-1B 5B 32 33 3B 33 37 6D
182Ø  1B 5B 3E 35 68 24 2Ø 2Ø-2Ø 2Ø 4B 49 4C 4C 2Ø 83
183Ø  47 83 89 81 5B 81 49 81-49 8E CØ 8D 73 82 FØ 83
184Ø  4C 83 83 83 93 83 5A 83-8B 82 B5 82 DC 82 B5 82
185Ø  BD A1 24 2Ø 2Ø 2Ø 2Ø 83-74 83 4Ø 83 43 83 8B 82

186Ø  AA 97 4C 82 E8 82 DC 82-B9 82 F1 2Ø 21 21 24 Ø7
187Ø  1B 5B 33 3B 31 48 1B 5B-3Ø 4A 1B 5B 32 33 3B 33
188Ø  37 6D 83 74 83 4Ø 83 43-83 8B 82 AA 97 4C 82 E8
189Ø  82 DC 82 B9 82 F1 24 Ø7-1B 5B 3E 35 68 1B 5B 32
18AØ  33 3B 33 37 6D 2A 2Ø 83-66 83 42 83 58 83 4E 82
18BØ  AA 83 5A 83 62 83 67 82-B3 82 EA 82 C4 82 A2 82
18CØ  DC 82 B9 82 F1 81 42 24-1B 5B 34 3B 31 48 1B 5B
18DØ  31 4D 1B 5B 31 33 3B 31-48 24 1B 5B 31 33 3B 31
18EØ  48 1B 5B 3Ø 4B 1B 5B 34-3B 31 48 1B 5B 31 4C 24
18FØ  1B 5B 3E 35 6C 24 1B 5B-3E 35 68 24 1B 5B 3Ø 34

19ØØ  3B 3Ø 31 48 24 24 1B 5B-32 31 3B 33 33 6D 54 4F
191Ø  2Ø 4D 53 2Ø 44 4F 53 1B-5B 32 33 3B 33 37 6D Ø7
192Ø  24 1B 5B 32 33 3B 34 3Ø-6D 24 1B 5B 32 33 3B 33
193Ø  37 6D 24 ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
194Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
195Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
196Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
197Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
198Ø  ØØ ØØ 2Ø Ø4 Ø4 Ø1 Ø1 1Ø-Ø7 ØØ ØØ ØØ ØØ ØØ ØØ Ø1
199Ø  ØØ ØØ ØØ ØØ Ø1 Ø1 ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ FA

19AØ  5Ø 53 51 52 1E Ø6 55 8C-C8 8E D8 C6 Ø6 DA 17 Ø1
19BØ  AØ D9 17 ØA CØ 74 12 AØ-D8 17 FE C8 A2 D8 17 75
19CØ  Ø8 BØ ØA A2 D8 17 E8 12-ØØ BØ 2Ø E6 ØØ E6 64 5D
19DØ  Ø7 1F 5A 59 5B 58 FB CF-ØØ ØØ ØØ E8 1E 1E 1E B8
19EØ  ØØ A8 8E D8 3Ø ØF B8 ØØ-BØ 8E D8 3Ø ØF B8 ØØ B8
19FØ  8E D8 3Ø ØF 2E 8Ø 3E 9E-17 Ø1 75 Ø7 B8 ØØ EØ 8E
1AØØ  D8 3Ø ØF E8 BE 1D 83 EB-5Ø B8 ØØ A8 E8 27 ØØ B8
1A1Ø  ØØ BØ E8 21 ØØ B8 ØØ B8-E8 1B ØØ 2E 8Ø 3E 9E 17
1A2Ø  Ø1 75 Ø6 B8 ØØ EØ E8 ØD-ØØ 1F 2E AØ 35 18 34 Ø1
1A3Ø  2E A2 35 18 C3 ØØ 8E D8-53 8Ø 37 FF 83 C3 5Ø B9

1A4Ø  Ø7 ØØ 8Ø 37 81 83 C3 5Ø-E2 F8 8Ø 37 FF 5B C3 E8
1A5Ø  AA 1D B6 7E AØ 88 17 3A-Ø6 87 17 75 Ø5 B6 42 AØ
1A6Ø  D3 1F 8A DØ 8A C1 F6 DØ-8A E8 52 51 1E BD ØØ A8
1A7Ø  E8 5B ØØ BD ØØ BØ E8 55-ØØ BD ØØ B8 E8 4F ØØ 2E
1A8Ø  8Ø 3E 9E 17 Ø1 75 Ø6 BD-ØØ EØ E8 41 ØØ 1F E8 53
1A9Ø  1D 59 5A 8A Ø7 22 C5 F6-C6 Ø4 75 Ø2 ØA C1 88 Ø7
1AAØ  52 E8 2Ø 1D 5A B9 5Ø ØØ-1E BD ØØ A8 E8 2E ØØ BD
1ABØ  ØØ BØ E8 28 ØØ BD ØØ B8-E8 22 ØØ 2E 8Ø 3E 9E 17
1ACØ  Ø1 75 Ø6 BD ØØ EØ E8 14-ØØ 1F E8 17 1D C3 8E DD
1ADØ  8A Ø7 22 C5 DØ CA 73 Ø2-ØA C1 88 Ø7 C3 8E DD 53

1AEØ  DØ CA 72 ØD BØ Ø7 C6 Ø7-ØØ Ø3 D9 FE C8 75 F7 5B
1AFØ  C3 C6 Ø7 7E BØ Ø5 Ø3 D9-88 37 FE C8 75 F8 Ø3 D9
1BØØ  C6 Ø7 7E 5B C3 BD ØØ A8-E8 1C ØØ BD ØØ BØ E8 16
1B1Ø  ØØ BD ØØ B8 E8 1Ø ØØ 2E-8Ø 3E 9E 17 Ø1 75 Ø6 BD
1B2Ø  ØØ EØ E8 Ø2 ØØ C3 ØØ 2E-8A ØE 26 19 1E 8E DD 2E
1B3Ø  AØ 95 17 8A E8 53 52 BF-96 17 51 53 52 87 FA 8A
1B4Ø  Ø5 87 FA 2E 22 Ø5 8A C8-2E 8A Ø5 F6 DØ 22 Ø7 ØA
1B5Ø  C1 8A C8 8A Ø7 88 ØF 87-D3 2E 22 Ø5 8A C8 2E 8A
1B6Ø  Ø5 F6 DØ 22 Ø7 ØA C1 88-Ø7 87 D3 43 42 47 FE CD
1B7Ø  75 CB 5A 5B 2E 8B ØE 88-19 2B D9 Ø3 D1 59 FE C9

1B8Ø  74 Ø2 EB B3 5A 5B 1F C3-ØØ ØØ BD ØØ A8 E8 1B ØØ
1B9Ø  BD ØØ BØ E8 15 ØØ BD ØØ-B8 E8 ØF ØØ 2E 8Ø 3E 9E
1BAØ  17 Ø1 75 Ø6 BD ØØ EØ E8-Ø1 ØØ C3 53 1E 8E DD 2E
1BBØ  8B ØE 94 17 8A C5 FE C8-74 Ø3 42 EB F9 51 52 53
1BCØ  8A CD 87 FA 32 CØ B5 Ø8-DØ ØF DØ DØ FE CD 75 F8
1BDØ  2E 88 Ø5 43 4F FE C9 75-EB 87 FA 5B 2E AØ F2 19
1BEØ  Ø2 C3 73 Ø2 FE C7 8A D8-5A 42 59 FE C9 75 C5 1F
1BFØ  5B C3 ØØ BD ØØ A8 E8 1B-ØØ BD ØØ BØ E8 15 ØØ BD
1CØØ  ØØ B8 E8 ØF ØØ 2E 8Ø 3E-9E 17 Ø1 75 Ø6 BD ØØ EØ
1C1Ø  E8 Ø1 ØØ C3 53 1E 8E DD-2E 8B ØE 94 17 87 FA 51

1C2Ø  53 8A Ø7 2E 88 Ø5 43 47-FE CD 75 F5 5B B1 5Ø Ø3
1C3Ø  D9 59 FE C9 75 E9 87 FA-1F 5B C3 BD ØØ A8 E8 1B
1C4Ø  ØØ BD ØØ BØ E8 15 ØØ BD-ØØ B8 E8 ØF ØØ 2E 8Ø 3E
1C5Ø  9E 17 Ø1 75 Ø6 BD ØØ EØ-E8 Ø1 ØØ C3 53 1E 8E DD
1C6Ø  2E 8B ØE 94 17 51 53 2E-8B 16 9B 1A 87 FA 2E 8A
```

```
1C7Ø  Ø5 87 FA F6 DØ 22 Ø7 88-Ø7 87 FA 2E 8A Ø5 87 FA
1C8Ø  2E 22 Ø5 ØA Ø7 88 Ø7 43-42 47 FE CD 75 DE 5B B1
1C9Ø  5Ø Ø3 D9 59 FE C9 75 CD-1F 5B C3 96 17 BD ØØ A8
1CAØ  E8 1B ØØ BD ØØ BØ E8 15-ØØ BD ØØ B8 E8 ØF ØØ 2E
1CBØ  8Ø 3E 9E 17 Ø1 75 Ø6 BD-ØØ EØ E8 Ø1 ØØ C3 B9 Ø8

1CCØ  C8 DØ C8 72 Ø3 B9 9Ø 9Ø-1E 5Ø 8E DD BD E7 1A 2E
1CDØ  89 4E ØØ 53 2E 8B ØE 94-17 51 53 BF 96 17 2E 8A
1CEØ  Ø5 8A C8 F6 DØ 22 Ø7 ØA-C1 88 Ø7 43 47 FE CD 75
1CFØ  ED 5B B1 5Ø Ø3 D9 59 FE-C9 75 DE 5B 58 1F C3 BD
1DØØ  ØØ A8 E8 1B ØØ BD ØØ BØ-E8 15 ØØ BD ØØ B8 E8 ØF
1D1Ø  ØØ 2E 8Ø 3E 9E 17 Ø1 75-Ø6 BD ØØ EØ E8 Ø1 ØØ C3
1D2Ø  8B ØE 82 17 1E 8E DD 53-51 53 8A Ø7 87 FA 2E 88
1D3Ø  Ø5 87 FA 43 42 FE CD 75-F1 5B B9 5Ø ØØ Ø3 D9 59
1D4Ø  FE C9 75 E4 5B 1F C3 BD-ØØ A8 E8 1B ØØ BD ØØ BØ
1D5Ø  E8 15 ØØ BD ØØ B8 E8 ØF-ØØ 2E 8Ø 3E 9E 17 Ø1 75

1D6Ø  Ø6 BD ØØ EØ E8 Ø1 ØØ C3-2E 8B ØE 82 17 1E 8E DD
1D7Ø  53 51 53 87 FA 2E 8A Ø5-87 FA 88 Ø7 43 42 FE CD
1D8Ø  75 F1 5B B9 5Ø ØØ Ø3 D9-59 FE C9 75 E4 5B 1F C3
1D9Ø  BD ØØ A8 E8 1D ØØ BD ØØ-BØ E8 17 ØØ BD ØØ B8 E8
1DAØ  11 ØØ 2E 8Ø 3E 9E 17 Ø1-75 Ø6 BD ØØ EØ E8 Ø3 ØØ
1DBØ  C3 ØØ ØØ 1E 8E DD E8 46-ØØ 2E 8B 1E B1 1B E8 3E
1DCØ  ØØ 2E 8B ØE 82 17 53 51-53 E8 33 ØØ 53 E8 2F ØØ
1DDØ  2E A1 B8 25 FF DØ 87 FA-2E 88 Ø5 87 FA 43 42 FE
1DEØ  CD 75 ED E8 19 ØØ 5B 83-C3 Ø6 E8 12 ØØ 5B 83 C3
1DFØ  5Ø 59 FE C9 75 D1 5B 1F-C3 ØØ ØØ ØØ ØØ ØØ ØØ 2E

1EØØ  87 1E F9 1B 2E 87 ØE FB-1B 2E 87 16 FD 1B C3 8A
1E1Ø  Ø7 C3 E8 EA FF 2E 8A Ø7-F6 DØ 43 E8 E1 FF 22 Ø7
1E2Ø  C3 BB C5 65 BA 5Ø ØØ 32-CØ 8A 26 9E 17 8Ø FC Ø1
1E3Ø  75 Ø5 B4 1Ø EB Ø3 9Ø B4-Ø8 B5 Ø8 53 E8 2F ØØ 5B
1E4Ø  43 FE CØ 3A C4 75 F2 B4-ØØ BB C5 65 Ø3 D8 AØ D3
1E5Ø  1F B5 Ø1 E8 18 ØØ B9 42-Ø6 E8 14 ØØ B5 Ø1 E8 ØD
1E6Ø  ØØ C3 8E DD C6 Ø7 ØØ 84-C2 74 Ø2 88 ØF C3 B1 7E
1E7Ø  1E BD ØØ A8 B2 Ø1 E8 E9-FF BD ØØ BØ B2 Ø2 E8 E1
1E8Ø  FF BD ØØ B8 B2 Ø4 E8 D9-FF 2E 8Ø 3E 9E 17 Ø1 75
1E9Ø  Ø8 BD ØØ EØ B2 Ø8 E8 C9-FF 83 C3 5Ø FE CD 75 D1

1EAØ  1F C3 8B 1E 89 17 BE 32-32 89 36 89 3B 89 36 B9
1EBØ  23 BE 35 35 89 36 8C 3B-BE 35 38 E8 D7 18 E8 9C
1ECØ  Ø6 BØ Ø1 E8 12 ØØ BE 32-32 89 36 B9 23 BE 35 38
1EDØ  AØ 86 17 E9 C2 Ø6 32 CØ-A2 4E 1D E8 A2 19 BA AF
1EEØ  FF Ø3 DA 53 E8 2D ØØ 89-1E FE 1C 5B B1 Ø3 E8 ØF
1EFØ  ØØ E8 2Ø ØØ 8B 1E FE 1C-B1 CØ E8 Ø3 ØØ C3 ØØ ØØ
1FØØ  AØ 82 17 8A E8 BA 5Ø ØØ-Ø3 DA E8 1E ØØ Ø3 DA FE
1F1Ø  CD 75 F7 C3 B1 Ø3 E8 12-ØØ 43 B1 FF AØ 83 17 8A
1F2Ø  E8 E8 Ø7 ØØ 43 FE CD 75-F8 B1 CØ 1E BD ØØ A8 E8
1F3Ø  1D ØØ BD ØØ BØ E8 17 ØØ-BD ØØ B8 E8 11 ØØ 2E 8Ø

1F4Ø  3E 9E 17 Ø1 75 Ø6 BD ØØ-EØ E8 Ø3 ØØ 1F C3 ØØ 8E
1F5Ø  DD 2E AØ 4E 1D ØA CØ 8A-C1 74 Ø5 ØA Ø7 88 Ø7 C3
1F6Ø  F6 DØ 22 Ø7 88 Ø7 C3 31-36 E8 14 19 1E BD ØØ A8
1F7Ø  E8 D8 ØØ BD ØØ BØ E8 D2-ØØ BD ØØ B8 E8 CC ØØ 2E
1F8Ø  8Ø 3E 9E 17 Ø1 75 Ø6 BD-ØØ EØ E8 BE ØØ 1F AØ 84
1F9Ø  17 3C Ø3 75 Ø1 C3 FF 16-67 1D AØ D3 1F A2 FF 1D
1FAØ  E8 5C FE BB 81 Ø2 89 1E-FD 1D BA 8Ø Ø2 E8 4F FE
1FBØ  AØ 82 17 8A DØ AØ 83 17-8A C8 53 B5 Ø8 DØ Ø7 E8
1FCØ  3D FE 73 13 53 E8 38 ØØ-B1 42 B5 Ø5 E8 33 ØØ FE
1FDØ  CD 75 F9 E8 2A ØØ 5B 43-E8 24 FE FE CD 75 DE 43

1FEØ  FE C9 75 D7 5B 83 C3 Ø6-E8 14 FE 8B 1E FD 1D Ø3
1FFØ  DA 89 1E FD 1D E8 Ø7 FE-FE CA 75 B9 C3 ØØ ØØ ØØ
2ØØØ  B1 7E AØ FF 1D 1E BD ØØ-A8 8E DD C6 Ø7 ØØ DØ C8
2Ø1Ø  73 Ø2 88 ØF BD ØØ BØ 8E-DD C6 Ø7 ØØ DØ C8 73 Ø2
2Ø2Ø  88 ØF BD ØØ B8 8E DD C6-Ø7 ØØ DØ C8 73 Ø2 88 ØF
2Ø3Ø  2E 8Ø 3E 9E 17 Ø1 75 ØE-BD ØØ EØ 8E DD C6 Ø7 ØØ
2Ø4Ø  DØ C8 73 Ø2 88 ØF 1F 83-C3 5Ø C3 53 8E DD E8 AE
2Ø5Ø  FD BB 81 Ø2 2E 89 1E B5-1E BA 5Ø ØØ E8 AØ FD 2E
2Ø6Ø  AØ 82 17 8A DØ 2E AØ 83-17 8A C8 53 B5 Ø8 8A Ø7
2Ø7Ø  E8 8C FD B1 7E DØ CØ 72-Ø2 B1 ØØ 53 B5 Ø7 88 ØF

2Ø8Ø  Ø3 DA FE CD 75 F8 5B 43-E8 74 FD FE CD 75 E1 43
2Ø9Ø  FE C9 75 D8 5B B9 5Ø ØØ-Ø3 D9 E8 62 FD 2E 8B 1E
2ØAØ  B5 1E B9 8Ø Ø2 Ø3 D9 2E-89 1E B5 1E E8 5Ø FD FE
2ØBØ  CA 75 B2 5B C3 ØØ ØØ BA-4A ØØ 1E BD ØØ A8 E8 1C
2ØCØ  ØØ BD ØØ BØ E8 16 ØØ BD-ØØ B8 E8 1Ø ØØ 2E 8Ø 3E
2ØDØ  9E 17 Ø1 75 Ø6 BD ØØ EØ-E8 Ø2 ØØ 1F C3 53 8E DD
2ØEØ  B1 18 B5 Ø6 C6 Ø7 ØØ 43-FE CD 75 F8 Ø3 DA FE C9
2ØFØ  75 FØ 5B C3 ØØ ØØ ØØ ØØ-34 36 E8 48 17 53 E8 82
21ØØ  17 89 1E F4 1E E8 AF FF-5A 8A E8 AØ 84 17 3C Ø3
211Ø  74 44 8A C5 53 52 FF 16-F8 1E 89 1E F6 1E 5A AØ

212Ø  85 17 FE C8 B8 9Ø 9Ø 74-Ø3 B8 F6 DØ BD 4Ø 1F 2E
213Ø  89 46 ØØ 8B ØE 82 17 EB-Ø1 9Ø 51 53 87 FA 8A Ø5

214Ø  9Ø 9Ø 88 Ø7 47 43 FE CD-75 F4 87 FA 5B 83 C3 Ø6
215Ø  59 FE C9 75 E5 5B 1E BD-ØØ A8 E8 8D ØØ BD ØØ BØ
216Ø  E8 87 ØØ BD ØØ B8 E8 81-ØØ 2E 8Ø 3E 9E 17 Ø1 75
217Ø  Ø6 BD ØØ EØ E8 73 ØØ 1F-AØ 84 17 3C Ø3 75 Ø1 C3
218Ø  8B 1E F4 1E 8B 16 F6 1E-8B ØE 82 17 51 53 52 87
219Ø  FA 8A Ø5 87 FA 8A ØE D3-1F 1E BD ØØ A8 E8 34 ØØ
21AØ  BD ØØ BØ E8 2E ØØ BD ØØ-B8 E8 28 ØØ 2E 8Ø 3E 9E
21BØ  17 Ø1 75 Ø6 BD ØØ EØ E8-1A ØØ 1F 42 43 FE CD 75

21CØ  CE 5A 83 C2 Ø6 5B B9 5Ø-ØØ Ø3 D9 59 FE C9 74 Ø2
21DØ  EB BA C3 Ø4 8E DD 5Ø F6-DØ 22 Ø7 88 Ø7 58 DØ C9
21EØ  72 Ø1 C3 5Ø ØA Ø7 88 Ø7-58 C3 8E DD 2E 8B ØE 82
21FØ  17 53 51 53 87 FA 2E 8A-Ø5 87 FA 88 Ø7 42 43 FE
22ØØ  CD 75 F1 5B B9 5Ø ØØ Ø3-D9 59 FE C9 75 E4 5B C3
221Ø  1E E8 Ø5 Ø1 BD ØØ A8 E8-1C ØØ BD ØØ BØ E8 16 ØØ
222Ø  BD ØØ B8 E8 1Ø ØØ 2E 8Ø-3E 9E 17 Ø1 75 Ø6 BD ØØ
223Ø  EØ E8 Ø2 ØØ 1F C3 8E DD-BB FØ ØØ 2E AØ 83 17 DØ
224Ø  CØ DØ CØ DØ CØ 8A C8 E8-AF ØØ FE C1 BA 5Ø ØØ 87
225Ø  D3 ØA CØ 1B D9 87 D3 2E-AØ 82 17 B4 ØØ D1 CØ D1

226Ø  CØ D1 CØ 4Ø C6 Ø7 1E Ø3-D9 C6 Ø7 78 Ø3 DA 48 75
227Ø  F3 FE C9 E8 83 ØØ 2E AØ-82 17 8A C8 FE C9 75 Ø1
228Ø  C3 BB B8 Ø4 E8 78 FB BB-31 16 BA ØØ 14 E8 6F FB
229Ø  E8 3E ØØ E8 3B ØØ E8 38-ØØ E8 35 ØØ E8 32 ØØ E8
22AØ  2F ØØ E8 2C ØØ Ø3 DA E8-55 FB 2E AØ 83 17 DØ CØ
22BØ  DØ CØ DØ CØ 2C Ø2 53 C6-Ø7 Ø1 43 C6 Ø7 81 FE C8
22CØ  75 F8 43 C6 Ø7 8Ø 5B Ø3-DA E8 33 FB FE C9 75 CØ
22DØ  C3 53 2E AØ 83 17 FE C8-74 12 8A E8 C6 Ø7 Ø1 43
22EØ  C6 Ø7 8Ø BA Ø7 ØØ Ø3 DA-FE CD 75 FØ 5B BA 8Ø Ø2
22FØ  Ø3 DA FE C9 74 Ø1 C3 58-C3 BØ Ø4 8A E9 53 C6 Ø7

23ØØ  1F B6 FF 43 88 37 FE CD-75 F9 43 C6 Ø7 F8 5B BA
231Ø  5Ø ØØ Ø3 DA FE C8 75 E3-C3 Ø6 BD ØØ A8 E8 1C ØØ
232Ø  BD ØØ BØ E8 16 ØØ BD ØØ-B8 E8 1Ø ØØ 2E 8Ø 3E 9E
233Ø  17 Ø1 75 Ø6 BD ØØ EØ E8-Ø2 ØØ Ø7 C3 8E C5 BE 9Ø
234Ø  Ø1 B8 ØØ ØØ 8B E8 8B FD-B9 19 ØØ F3 AB 83 C5 5Ø
235Ø  4E 75 F3 C3 1E BD ØØ A8-E8 1C ØØ BD ØØ BØ E8 16
236Ø  ØØ BD ØØ B8 E8 1Ø ØØ 2E-8Ø 3E 9E 17 Ø1 75 Ø6 BD
237Ø  ØØ EØ E8 Ø2 ØØ 1F C3 8E-DD 8E C5 BE 52 2B BØ 7C
238Ø  C7 Ø4 ØØ ØØ 8B FE 47 B9-ØE ØØ F3 A5 A4 83 C6 33
239Ø  FE C8 75 EC C3 Ø6 BA 9F-17 B8 ØA 35 CD 21 89 1E

23AØ  E6 ØØ 8C Ø6 E8 ØØ Ø7 B8-ØA 25 CD 21 FA E4 Ø2 A2
23BØ  EA ØØ 24 FB E6 Ø2 FB E6-64 E8 73 ØØ E8 9E Ø4 E8
23CØ  14 FB E8 2Ø ØØ BA 8F 13-E8 Ø3 DF BØ ØØ A2 A8 12
23DØ  1E 2E C5 16 E6 ØØ B8 ØA-25 CD 21 1F FA 2E AØ EA
23EØ  ØØ E6 Ø2 FB C3 BF ØØ ØØ-B9 9Ø Ø1 B8 ØØ A8 E8 2Ø
23FØ  ØØ B8 ØØ BØ E8 1A ØØ B8-ØØ B8 E8 14 ØØ 2E 8Ø 3E
24ØØ  9E 17 Ø1 75 Ø6 B8 ØØ EØ-E8 Ø6 ØØ 83 C7 5Ø E2 DB
241Ø  C3 83 F9 7E 7D Ø6 BB 32-ØØ EB Ø4 9Ø BB 1A ØØ 51
242Ø  57 Ø6 8E CØ B8 ØØ ØØ 8B-CB F3 AB Ø7 5F 59 C3 E8
243Ø  34 12 E8 71 14 E8 67 14-E8 E6 F9 AØ 88 17 E8 2F

244Ø  12 E8 85 1Ø E8 F7 12 E8-C6 FD E8 55 FA E8 19 FB
245Ø  E8 D7 Ø5 BA F6 16 E8 75-DE E6 64 B8 Ø7 ØC CD 21
246Ø  BB 53 22 53 3C 3Ø 7C ØC-3C 39 7F Ø8 8A C8 8Ø E1
247Ø  ØF E9 A3 Ø4 3C 2B 75 Ø3-E9 F6 Ø3 3C 3D 75 Ø3 E9
248Ø  16 Ø4 3C 2E 75 Ø3 E9 46-Ø4 3C ØD 75 Ø3 E9 51 Ø2
249Ø  3C 1B 75 Ø2 58 C3 3C 2Ø-75 Ø3 E9 7C Ø5 3C 43 75
24AØ  Ø3 E9 47 11 3C 63 75 Ø3-E9 4Ø 11 3C 44 75 Ø3 E9
24BØ  8D ØE 3C 64 75 Ø3 E9 86-ØE 3C 46 75 Ø3 E9 8C Ø6
24CØ  3C 66 75 Ø3 E9 85 Ø6 3C-6D 75 Ø3 E9 CB Ø5 3C 4D
24DØ  75 Ø3 E9 C4 Ø5 3C 7Ø 75-Ø3 E9 28 ØD 3C 5Ø 75 Ø3

24EØ  E9 21 ØD 3C 53 75 Ø3 E9-D4 ØE 3C 73 75 Ø3 E9 CD
24FØ  ØE 3C 74 75 Ø3 E9 DØ Ø6-3C 54 75 Ø3 E9 C9 Ø6 3C
25ØØ  58 75 Ø3 EB 43 9Ø 3C 78-75 Ø3 EB 3C 9Ø 3C 5A 75
251Ø  Ø3 EB 2A 9Ø 3C 7A 74 25-B8 ØE Ø4 CD 18 F6 C4 1Ø
252Ø  74 Ø3 E9 9C ØØ F6 C4 Ø8-74 Ø3 E9 D8 Ø4 B8 Ø7 Ø4
253Ø  CD 18 F6 C4 4Ø 74 Ø3 E9-88 Ø3 E9 C1 Ø4 8B 1E 89
254Ø  17 89 1E 8B 17 EB Ø9 9Ø-8B 1E 89 17 89 1E 8D 17
255Ø  AØ 86 17 A2 8F 17 E8 Ø4-ØØ E8 57 ØF C3 8B 1E 8B
256Ø  17 BE 32 33 89 36 89 3B-89 36 B9 23 BE 35 35 89
257Ø  36 8C 3B BE 35 38 E8 19-ØØ 8B 1E 8D 17 BE 32 34

258Ø  89 36 89 3B 89 36 B9 23-BE 35 35 89 36 8C 3B BE
259Ø  35 38 E8 ØØ 12 AØ 8F 17-BB 8Ø 38 8A DØ B6 ØØ Ø3
25AØ  DA 8A Ø7 A2 BF 23 BA B7-23 E8 22 DD 89 36 8C 3B
25BØ  BA 83 3B E8 18 DD C3 1B-5B 32 34 3B 36 32 48 31
25CØ  24 B8 Ø2 Ø4 CD 18 F6 C4-Ø8 74 Ø3 E9 Ø2 Ø2 F6 C4
25DØ  2Ø 74 Ø3 EB 2E 9Ø B8 Ø3-Ø4 CD 18 F6 C4 2Ø 74 Ø3
25EØ  E9 D7 Ø1 F6 C4 8Ø 74 Ø3-E9 C6 ØØ B8 Ø4 Ø4 CD 18
25FØ  F6 C4 Ø1 74 Ø3 E9 C5 ØØ-B8 Ø5 Ø4 CD 18 F6 C4 Ø4
26ØØ  75 4E C3 E8 57 Ø2 E8 DE-13 AØ 94 17 DØ E8 75 Ø3
261Ø  E9 9D ØE A2 26 19 BØ Ø6-A2 88 19 E8 BC 11 BØ Ø6
```

```
262Ø   E8 1B ØØ 8C CD E8 FF F4-BØ 5Ø A2 88 19 E8 C1 11
263Ø   BØ 5Ø E8 Ø9 ØØ E8 CD F4-E8 72 ØE E9 1E FF 53 8A
264Ø   DØ B6 ØØ AØ 94 17 FE C8-74 Ø4 Ø3 DA EB F8 5A C3
265Ø   E8 ØA Ø2 E8 91 13 E8 81-11 BA ØØ 85 BØ Ø6 A2 F2
266Ø   19 8C CD E8 45 F5 52 E8-87 11 5A BØ 5Ø A2 F2 19
267Ø   E8 17 F5 32 CØ 87 FA 88-Ø5 87 FA BF ØØ 85 AØ 92
268Ø   17 24 Ø7 FE CØ 8A C8 AØ-9Ø 17 24 Ø7 Ø2 C1 2C Ø8
269Ø   73 Ø3 Ø4 Ø8 47 A2 BØ 24-E8 4E Ø4 BB 96 17 89 1E
26AØ   9B 1A E8 35 11 E8 71 Ø4-E8 46 11 E8 8D F5 EB 88
26BØ   ØØ E8 95 Ø1 E8 A9 ØØ E8-F9 ØD EB 1C 9Ø E8 89 Ø1

26CØ   AØ 86 17 5Ø BØ Ø1 5Ø A2-86 17 E8 93 ØØ 58 FE CØ
26DØ   3C ØD 75 F2 58 A2 86 17-E8 D5 ØD E8 Ø3 ØØ E9 7B
26EØ   FE 2E 8Ø 3E 9E 17 Ø1 74-1D BØ 37 A2 6E 38 E6 A8
26FØ   BØ 15 A2 6F 38 E6 AA BØ-26 A2 7Ø 38 E6 AC BØ Ø4
27ØØ   A2 71 38 E6 AE C3 57 51-B9 1Ø ØØ BF 3Ø 25 B4 ØØ
271Ø   8A C4 E6 A8 8A 85 ØØ ØØ-E6 AE 8A 85 Ø1 ØØ E6 AC
272Ø   8A 85 Ø2 ØØ E6 AA 83 C7-Ø3 FE C4 E2 E3 59 5F C3
273Ø   ØØ ØØ ØØ ØF ØØ ØØ ØØ ØF-ØØ ØF ØF ØØ ØØ ØØ ØF ØF
274Ø   ØØ ØF ØØ ØF ØF ØF ØF ØF-Ø7 Ø7 Ø7 ØA ØØ ØØ ØØ ØA
275Ø   ØØ ØA ØA ØØ ØØ ØØ ØA ØA-ØØ ØA ØØ ØA ØA ØA ØA ØA

276Ø   AØ 85 17 FE C8 B8 9Ø 9Ø-74 Ø3 B8 F6 DØ BD 94 25
277Ø   2E 89 46 ØØ E8 BA 1Ø 89-1E B1 1B 53 E8 C6 1Ø 5A
278Ø   B9 ØF 1C AØ 84 17 3C Ø3-74 22 8B ØE 82 17 51 52
279Ø   87 FA 8A Ø5 9Ø 9Ø 88 Ø7-47 43 FE CD 75 F4 87 FA
27AØ   5A 83 C2 Ø6 59 FE C9 75-E5 B9 12 1C 89 ØE B8 25
27BØ   53 E8 CC 1Ø 5A E9 D8 F5-12 1C E8 AØ ØØ E8 16 F7
27CØ   BØ Ø1 5Ø E8 34 F9 58 FE-CØ 3C ØD 75 F5 EB ØD 9Ø
27DØ   E8 8A ØØ E8 ØØ F7 AØ 86-17 E8 1E F9 E8 E2 F6 E9
27EØ   56 FE 56 57 BE 17 26 BF-21 26 B9 Ø5 ØØ B8 Ø8 Ø4
27FØ   CD 18 8A C4 84 Ø4 75 19-46 47 E2 F8 B9 Ø5 ØØ B8

28ØØ   Ø9 Ø4 CD 18 8A C4 84 Ø4-75 Ø7 46 47 E2 F8 F9 EB
281Ø   Ø3 8A ØD F8 5F 5E C3 Ø4-Ø8 1Ø 4Ø 8Ø Ø1 Ø4 Ø8 1Ø
282Ø   4Ø Ø7 Ø8 Ø9 Ø4 Ø5 Ø6 Ø1-Ø2 Ø3 ØØ 51 B9 ØF ØØ B4
283Ø   Ø1 8A C4 E6 A8 BØ ØF E6-AA BØ ØF E6 AC BØ ØØ E6
284Ø   AE FE C4 E2 EC 59 EB 15-9Ø 2E 8Ø 3E 9E 17 Ø1 74
285Ø   DA BØ 66 E6 A8 E6 AA E6-AC BØ Ø6 E6 AE FA 2E AØ
286Ø   35 18 ØA CØ 74 Ø3 E8 72-F1 32 CØ 2E A2 D9 17 FB
287Ø   C3 E8 E9 FF E8 BA ØF BA-6Ø 7F B9 2Ø Ø1 87 F3 87
288Ø   FA 8C C8 8E CØ FC F3 A4-87 F3 87 FA 52 E8 FØ ØF
289Ø   5A E8 6B F4 E8 19 ØC C3-E8 C2 FF E8 93 ØF BA 6Ø

28AØ   7F B9 2Ø Ø1 87 D3 87 F3-87 FA 8C C8 8E CØ FC F3
28BØ   A4 87 F3 87 FA 53 E8 C7-ØF 5A E8 8A F4 E8 ED ØB
28CØ   EB D5 E8 98 FF BB ØØ ØØ-89 1E 89 17 EB 1A 9Ø E8
28DØ   8B FF AØ 82 17 FE C8 A2-8A 17 AØ 83 17 DØ CØ DØ
28EØ   CØ DØ CØ FE C8 A2 89 17-E8 B5 ØØ E8 C5 ØB B8 ØA
28FØ   Ø4 CD 18 F6 C4 Ø1 75 F6-B8 Ø7 Ø4 CD 18 F6 C4 4Ø
29ØØ   75 EC C3 Ø5 28 27 27 2C-27 31 27 36 27 19 28 3B
291Ø   27 4Ø 27 45 27 4A 27 BB-Ø3 27 DØ E1 B5 ØØ Ø3 D9
292Ø   8B 17 43 87 D3 FF E3 B9-FF Ø1 EB 21 B9 ØØ Ø1 EB
293Ø   1C B9 Ø1 Ø1 EB 17 B9 FF-ØØ EB 12 B9 Ø1 ØØ EB ØD

294Ø   B9 FF FF EB Ø8 B9 ØØ FF-EB Ø3 B9 Ø1 FF 51 E8 ØC
295Ø   FF 59 B8 ØE Ø4 CD 18 F6-C4 Ø4 74 Ø3 E9 FE ØØ F6
296Ø   C4 Ø1 74 Ø8 DØ E5 DØ E5-DØ E1 DØ E1 8B 1E 82 17
297Ø   DØ E7 DØ E7 DØ E7 AØ 89-17 Ø2 C1 B1 ØØ 78 ØA 8A
298Ø   C8 3A C7 72 Ø4 8A CF FE-C9 AØ 8A 17 Ø2 C5 B5 ØØ
299Ø   78 ØA 8A E8 3A C3 72 Ø4-8A EB FE CD 89 ØE 89 17
29AØ   8B 1E 89 17 BE 32 32 89-36 89 3B 89 36 B9 23 BE
29BØ   35 35 89 36 8C 3B BE 35-38 E8 D9 ØD B8 Ø6 Ø4 CD
29CØ   18 F6 C4 1Ø 74 Ø3 E8 86-FØ B8 ØE Ø4 CD 18 F6 C4
29DØ   Ø8 74 Ø3 E8 76 ØØ E8 51-ØØ 8A 16 Ø4 28 E4 AØ A8

29EØ   2Ø 74 FA E8 FC FD 73 Ø8-ØA CØ 74 Ø4 3C 22 75 ØE
29FØ   E4 AØ A8 2Ø 75 FA FE CA-75 E3 BØ Ø2 EB Ø2 BØ 1E
2AØØ   A2 Ø4 28 C3 1E B8 ØE Ø4-CD 18 F6 C4 Ø4 74 Ø1 C3
2A1Ø   E8 4A FE E8 36 ØØ EB 12-9Ø B8 ØE Ø4 CD 18 F6 C4
2A2Ø   Ø4 74 Ø1 C3 E8 36 FE E8-25 FØ BØ Ø1 A2 D9 17 A2
2A3Ø   D8 17 C3 1B 5B 3Ø 32 3B-3Ø 31 48 1B 5B 3Ø 6D 1B
2A4Ø   29 33 83 1B 5B 3Ø 6D 1B-29 3Ø 24 24 AØ 88 17 5Ø
2A5Ø   32 CØ A2 88 17 E8 F7 EF-58 A2 88 17 C3 AØ 86 17
2A6Ø   Ø2 C1 DØ E5 DØ E5 Ø2 C5-FE C8 3C ØC 72 Ø2 EB BA
2A7Ø   FE CØ 5Ø E8 6Ø F4 58 A2-86 17 E8 44 F4 E8 2D ØA

2A8Ø   E8 A7 FF 8B ØE 82 17 32-CØ Ø2 C1 FE CD 75 FA 3C
2A9Ø   19 72 Ø1 C3 B2 Ø2 E9 44-FF E8 9D ØB E8 B5 ØE E8
2AAØ   38 ØD BA ØØ 85 8B ØE 94-17 51 53 8A Ø7 87 FA 88
2ABØ   Ø5 87 FA 43 42 FE CD 75-F2 5B 83 C3 Ø6 59 FE C9
2ACØ   74 Ø2 EB E5 52 E8 29 ØD-5A E8 27 F1 32 CØ 87 FA
2ADØ   88 Ø5 87 FA E8 12 ØØ E8-ØA ØD E8 3C ØØ E8 1C ØD
2AEØ   E8 58 F1 E8 83 F4 E9 3D-ØB A1 94 17 F6 E4 B9 Ø4
2AFØ   ØØ Ø2 ØE 9E 17 F7 E1 8B-D8 43 87 D3 8A 2E BØ 24

2BØØ   BB ØØ 85 FE CD 79 Ø1 C3-52 ØA CØ DØ 1F 43 FE CA
2B1Ø   75 F9 FE CE 79 F5 5A EB-E7 8B ØE 94 17 51 53 8B

2B2Ø   16 9B 1A 87 FA 8A Ø5 87-FA F6 DØ 22 Ø7 88 Ø7 87
2B3Ø   FA 8A Ø5 87 FA 22 Ø5 ØA-Ø7 88 Ø7 43 42 47 FE CD
2B4Ø   75 E1 5B 83 C3 Ø6 59 FE-C9 75 D2 C3 E8 EA ØA E8
2B5Ø   95 ØE E8 9C ØC AØ 88 17-3A Ø6 87 17 75 Ø3 AØ D3
2B6Ø   1F 5Ø 9C E8 37 F1 9D 58-B9 Ø8 C8 74 Ø3 B9 9Ø 9Ø
2B7Ø   BD 8F 29 2E 89 4E ØØ E8-6Ø ØC 8B ØE 94 17 51 53
2B8Ø   BA 96 17 87 FA 8A Ø5 87-FA 8A C8 F6 DØ 22 Ø7 ØA
2B9Ø   C1 88 Ø7 43 42 FE CD 75-EA 5B 83 C3 Ø6 59 FE C9
2BAØ   75 DC E8 C4 F3 E9 7E ØA-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
2BBØ   2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-31 32 33 34 35 36 37 38

2BCØ   39 41 42 43 44 45 46 ØØ-E8 92 FC E8 B2 ØC 89 1E
2BDØ   F4 1E BA AØ 8A E8 27 F1-E8 2D ØB 54 3D ØØ B8 36
2BEØ   34 A3 9Ø 34 B8 31 39 A3-8D 34 BA 8B 34 E8 DE D6
2BFØ   BF A8 29 B5 Ø1 E8 4Ø Ø2-3C ØD 75 Ø3 EB 5E 9Ø 3C
2CØØ   1B 75 Ø3 E9 23 ØA 3C Ø8-75 Ø3 EB 3A 9Ø 3C 31 7C
2C1Ø   E4 3C 39 7E 12 3C 41 7C-DC 3C 43 7E ØA 3C 61 7C
2C2Ø   D4 3C 63 7F DØ 24 DF 8A-CD FE C1 F6 C1 1Ø 74 Ø2
2C3Ø   EB C3 FE C5 88 Ø5 47 88-2E C7 29 8A DØ B4 Ø2 CD
2C4Ø   21 E8 51 ØA EB AF 8A C5-FE C8 75 Ø2 EB A7 FE CD
2C5Ø   4F E8 5D ØA BA BC 3B E8-74 D6 EB 99 FE CD 75 Ø3

2C6Ø   E9 C6 Ø9 E8 CC Ø9 E8 9F-ØA 57 41 49 54 28 31 7E
2C7Ø   39 29 3D ØØ E8 EF Ø9 B8-37 33 A3 9Ø 34 B8 31 39
2C8Ø   A3 8D 34 BA 8B 34 E8 45-D6 BA 92 3B E8 3F D6 AØ
2C9Ø   28 2C ØC 3Ø 52 8A DØ B4-Ø2 CD 21 5A BA 97 3B E8
2CAØ   2C D6 E8 88 Ø1 E8 9Ø Ø1-3C ØD 74 15 3C 3Ø 7E F5
2CBØ   3C 39 7F F1 52 8A DØ B4-Ø2 CD 21 5A 24 ØF A2 28
2CCØ   2C E8 6E Ø9 E8 9F Ø9 E8-E9 Ø7 E8 3B ØA 54 3D ØØ
2CDØ   B8 36 34 A3 9Ø 34 B8 31-39 A3 8D 34 BA 8B 34 E8
2CEØ   EC D5 8A 2E C7 29 FE CD-BF A8 29 8A 15 47 B4 Ø2
2CFØ   CD 21 FE CD 75 F5 C6 Ø5-2Ø E8 6A Ø9 E8 2E Ø1 BB

2DØØ   29 2C 89 1E F8 1E 89 1E-67 1D EB ØD 9Ø BA 97 3B
2D1Ø   E8 BB D5 8A 17 B4 Ø2 CD-21 B8 36 34 A3 9Ø 34 B8
2D2Ø   31 39 A3 8D 34 BA 8B 34-E8 A3 D5 BA 92 3B E8 9D
2D3Ø   D5 BB A8 29 8A 17 B4 Ø2-CD 21 BA 8B 34 E8 8E D5
2D4Ø   8A Ø7 3C 2Ø 74 C7 53 E8-B5 FØ BB C3 29 B9 ØC ØØ
2D5Ø   ØE Ø7 FD 87 DF F2 AE 87-DF FC 8A C1 FE CØ E8 9E
2D6Ø   FØ E8 E1 ØA 8B 16 F4 1E-87 D3 E8 9C F3 8A ØE 28
2D7Ø   2C AØ DA 17 22 CØ 74 F9-BØ ØØ A2 DA 17 B8 Ø3 Ø4
2D8Ø   CD 18 F6 C4 1Ø 74 Ø3 EB-74 9Ø B8 ØØ Ø4 CD 18 F6
2D9Ø   C4 Ø1 74 Ø3 EB 67 9Ø B8-Ø6 Ø4 CD 18 F6 C4 1Ø 74

2DAØ   2A E8 C5 F1 B8 Ø3 Ø4 CD-18 F6 C4 1Ø 74 Ø3 EB 4D
2DBØ   9Ø B8 ØØ Ø4 CD 18 F6 C4-Ø1 74 Ø3 EB 4Ø 9Ø B8 Ø6
2DCØ   Ø4 CD 18 F6 C4 1Ø 74 DC-EB 13 9Ø AØ DA 17 22 CØ
2DDØ   74 F9 BØ ØØ A2 DA 17 FE-C9 74 Ø2 EB 94 5B BA 97
2DEØ   3B E8 EA D4 8A 17 B4 Ø2-CD 21 43 BA 92 3B E8 DD
2DFØ   D4 8A 17 B4 Ø2 CD 21 E8-9B Ø8 E9 43 FF 5B BA 97
2EØØ   3B E8 CA D4 8A 17 B4 Ø2-CD 21 BB 34 36 89 1E F8
2E1Ø   1E BB 31 36 89 1E 67 1D-8B 1E F4 1E BA AØ 8A E8
2E2Ø   25 EF E8 44 F1 E9 11 Ø5-Ø4 BB ØØ 85 C3 B8 Ø3 Ø4
2E3Ø   CD 18 F6 C4 1Ø 75 F6 C3-B4 Ø7 CD 21 5Ø BA F6 16

2E4Ø   E8 8B D4 58 C3 BF 46 5A-C6 Ø6 BE 2E ØØ 9Ø C7 Ø6
2E5Ø   BF 2E ØA ØØ E8 AC Ø1 8B-36 C1 2E 8A ØC B5 ØØ 8A
2E6Ø   C1 3C ØA 7C Ø2 Ø4 Ø7 Ø4-3Ø A2 9C 39 BF 46 5A C6
2E7Ø   Ø6 28 2D Ø1 9Ø B8 ØØ A8-E8 AE ØØ 81 C7 9Ø Ø1 46
2E8Ø   8A ØC B5 ØØ 8A C1 3C ØA-7C Ø2 Ø4 Ø7 Ø4 3Ø A2 AE
2E9Ø   39 C6 Ø6 28 2D Ø2 9Ø B8-ØØ BØ E8 8C ØØ 81 C7 9Ø
2EAØ   Ø1 46 8A ØC B5 ØØ 8A C1-3C ØA 7C Ø2 Ø4 Ø7 Ø4 3Ø
2EBØ   A2 CØ 39 C6 Ø6 28 2D Ø4-9Ø B8 ØØ B8 E8 6A ØØ BF
2ECØ   47 5A AØ BD 2E 98 D1 E8-B9 9Ø Ø1 F7 E1 Ø3 F8 B8
2EDØ   ØØ A8 E8 13 ØØ B8 ØØ BØ-E8 ØD ØØ B8 ØØ B8 E8 Ø7

2EEØ   ØØ B8 ØØ EØ E8 Ø1 ØØ C3-57 Ø6 8E CØ B8 FF FF 26
2EFØ   Ø9 Ø5 26 Ø9 45 Ø2 26 Ø9-45 Ø4 26 81 4D Ø6 FF FØ
2FØØ   83 C7 5Ø B9 Ø4 ØØ 26 8Ø-ØD 8Ø 26 8Ø 4D Ø7 1Ø 83
2F1Ø   C7 5Ø E2 F2 26 Ø9 Ø5 26-Ø9 45 Ø2 26 Ø9 45 Ø4 26
2F2Ø   81 4D Ø6 FF FØ Ø7 5F C3-Ø1 56 57 33 CØ 8B DØ 8B
2F3Ø   D8 8B E8 23 C9 74 14 B8-ØØ FØ 49 E3 ØE D1 E1 D1
2F4Ø   E1 D1 F8 D1 DA D1 DB D1-DD E2 F6 86 C4 86 D6 86
2F5Ø   DF 87 DD 86 DF F6 Ø6 28-2D Ø1 74 Ø6 B9 ØØ A8 E8
2F6Ø   2Ø ØØ F6 Ø6 28 2D Ø2 74-Ø6 B9 ØØ BØ E8 13 ØØ F6
2F7Ø   Ø6 28 2D Ø4 74 Ø6 B9 ØØ-B8 E8 Ø6 ØØ E8 87 F7 5F

2F8Ø   5E C3 Ø6 8E C1 B9 Ø5 ØØ-26 C6 Ø5 1E 47 26 Ø9 Ø5
2F9Ø   26 Ø9 55 Ø2 26 Ø9 6D Ø4-26 Ø9 5D Ø6 F6 Ø6 28 2D
2FAØ   Ø2 74 Ø8 4F E8 ØB ØØ E8-Ø8 ØØ 47 83 C7 4F E2 D8
2FBØ   Ø7 C3 26 DØ 65 Ø8 26 DØ-55 Ø7 26 DØ 55 Ø6 26 DØ
2FCØ   55 Ø5 26 DØ 55 Ø4 26 DØ-55 Ø3 26 DØ 55 Ø2 26 DØ
```

```
2FDØ  55 Ø1 26 DØ 15 C3 BF 3D-5A C6 Ø6 BE 2E ØØ 9Ø C7
2FEØ  Ø6 BF 2E 12 ØØ E8 1B ØØ-C3 BF 3D 5A C6 Ø6 BE 2E
2FFØ  ØØ 9Ø C7 Ø6 BF 2E Ø2 ØØ-E8 Ø8 ØØ 8A ØE BC 2E 88
3ØØØ  ØE BE 2E AØ BE 2E 33 D2-DØ C8 83 DA ØØ B9 ØØ A8
3Ø1Ø  E8 28 ØØ 33 D2 DØ C8 83-DA ØØ B9 ØØ BØ E8 1B ØØ

3Ø2Ø  33 D2 DØ C8 83 DA ØØ B9-ØØ B8 E8 ØE ØØ 33 D2 DØ
3Ø3Ø  C8 83 DA ØØ B9 ØØ EØ E8-Ø1 ØØ C3 Ø6 57 8E C1 B9
3Ø4Ø  1Ø ØØ 57 51 8B ØE BF 2E-26 88 15 47 E2 FA 59 5F
3Ø5Ø  83 C7 5Ø E2 ED 5F Ø7 C3-57 51 B9 1Ø ØØ BF 8C 2E
3Ø6Ø  BE 3Ø 25 B4 ØØ 8A C4 E6-A8 8A Ø5 88 Ø4 E6 AE 8A
3Ø7Ø  45 Ø1 88 44 Ø1 E6 AC 8A-45 Ø2 88 44 Ø2 E6 AA 83
3Ø8Ø  C7 Ø3 83 C6 Ø3 FE C4 E2-DC 59 5F C3 ØØ ØØ ØØ ØF
3Ø9Ø  ØØ ØØ ØØ ØF ØØ ØF ØF ØØ-ØØ ØØ ØF ØF ØØ ØF ØØ ØF
3ØAØ  ØF ØF ØF ØF Ø7 Ø7 Ø7 ØA-ØØ ØØ ØØ ØA ØØ ØA ØA ØØ
3ØBØ  ØØ ØØ ØA ØA ØØ ØA ØØ ØA-ØA ØA ØA ØA Ø7 Ø1 ØØ Ø1

3ØCØ  ØØ 3Ø 25 9C 39 AE 39 CØ-39 E8 91 F7 AØ 88 17 3C
3ØDØ  ØA 7C Ø2 Ø4 Ø7 Ø4 3Ø A2-88 39 AØ 88 17 A2 BC 2E
3ØEØ  8A EØ Ø2 C4 Ø2 C4 B4 ØØ-Ø5 3Ø 25 A3 C1 2E E8 54
3ØFØ  FD BA 7F 39 E8 D7 D1 E8-EF FE E8 58 Ø4 3C 1B 74
31ØØ  Ø8 3C ØD 74 Ø4 3C 2Ø 75-12 E8 CA FE AØ BC 2E A2
311Ø  88 17 E8 5B Ø5 E8 9B Ø3-E9 ØE Ø5 3C 49 74 Ø4 3C
312Ø  69 75 ØC E8 32 FF E8 AD-FE E8 87 Ø3 E9 FA Ø4 3C
313Ø  38 75 2E AØ BC 2E FE C8-78 B7 A2 BC 2E 3C ØA 7C
314Ø  Ø2 Ø4 Ø7 Ø4 3Ø A2 88 39-AØ BC 2E 8A EØ Ø2 C4 Ø2
315Ø  C4 B4 ØØ Ø5 3Ø 25 A3 C1-2E E8 8D FE E8 E6 FC EB

316Ø  9Ø 3C 32 75 31 AØ BC 2E-FE CØ 3C 1Ø 74 83 A2 BC
317Ø  2E 3C ØA 7C Ø2 Ø4 Ø7 Ø4-3Ø A2 88 39 AØ BC 2E 8A
318Ø  EØ Ø2 C4 Ø2 C4 B4 ØØ Ø5-3Ø 25 A3 C1 2E E8 59 FE
319Ø  E8 B2 FC E9 5B FF 3C 34-75 1B 8B 3E C1 2E B4 ØØ
31AØ  AØ BD 2E D1 E8 Ø3 F8 8A-Ø5 FE C8 78 Ø5 88 Ø5 E8
31BØ  93 FC E9 3C FF 3C 36 75-1D 8B 3E C1 2E B4 ØØ AØ
31CØ  BD 2E D1 E8 Ø3 F8 8A Ø5-FE CØ 3C 1Ø 7D Ø5 88 Ø5
31DØ  E8 72 FC E9 1B FF 3C ØA-75 14 AØ BD 2E DØ EØ 3C
31EØ  Ø4 7E Ø2 BØ Ø1 A2 BD 2E-E8 5A FC E9 Ø3 FF 3C ØB
31FØ  75 ØF AØ BD 2E DØ E8 73-Ø2 BØ Ø4 A2 BD 2E E8 44

32ØØ  FC E9 ED FE 8Ø 3E 9E 17-Ø1 75 Ø3 E9 BB FE 9Ø E8
321Ø  F3 Ø4 5Ø 61 6C 6C 65 74-28 3Ø 7E 38 29 3D ØØ E8
322Ø  33 Ø3 3C 3Ø 73 Ø3 E9 ØØ-Ø4 3C 39 72 Ø3 E9 F9 Ø3
323Ø  8A DØ B4 Ø2 CD 21 24 ØF-8A C8 E8 76 Ø2 B2 1C B4
324Ø  Ø2 CD 21 E8 F2 FB 3C 3Ø-73 Ø3 E9 DC Ø3 3C 38 72
325Ø  Ø3 E9 D5 Ø3 8A DØ B4 Ø2-CD 21 24 ØF 8A E8 8A C1
326Ø  3C Ø8 75 Ø3 E9 A4 ØØ 3C-ØØ 75 13 AØ 71 38 B1 Ø4
327Ø  D2 E5 24 ØF ØA C5 A2 71-38 E6 AE E9 A8 Ø3 3C Ø1
328Ø  75 13 AØ 6F 38 B1 Ø4 D2-E5 24 ØF ØA C5 A2 6F 38
329Ø  E6 AA E9 91 Ø3 3C Ø2 75-13 AØ 7Ø 38 B1 Ø4 D2 E5

32AØ  24 ØF ØA C5 A2 7Ø 38 E6-AC E9 7A Ø3 3C Ø3 75 13
32BØ  AØ 6E 38 B1 Ø4 D2 E5 24-ØF ØA C5 A2 6E 38 E6 A8
32CØ  E9 63 Ø3 3C Ø4 75 ØF AØ-71 38 24 FØ ØA C5 A2 71
32DØ  38 E6 AE E9 5Ø Ø3 3C Ø5-75 ØF AØ 6F 38 24 FØ ØA
32EØ  C5 A2 6F 38 E6 AA E9 3D-Ø3 3C Ø6 75 ØF AØ 7Ø 38
32FØ  24 FØ ØA C5 A2 7Ø 38 E6-AC E9 2A Ø3 AØ 6E 38 24
33ØØ  FØ ØA C5 A2 6E 38 E6 A8-E9 1B Ø3 8A C5 A2 D3 1F
331Ø  AØ 86 17 5Ø BØ Ø1 5Ø A2-86 17 E8 66 Ø5 89 1E F4
332Ø  1E E8 1Ø Ø5 89 1E F6 1E-E8 4D EE 58 FE CØ 3C ØD
333Ø  75 E4 58 A2 86 17 E8 E8-EA E8 2D EC E9 E7 Ø2 E8

334Ø  F8 Ø1 E8 C3 Ø3 44 61 74-61 2Ø 63 68 61 6E 67 65
335Ø  2Ø 28 34 2D 36 29 ØØ E8-ØC Ø3 BA 92 3B E8 6E CF
336Ø  E8 8C Ø1 73 Ø3 EB 38 9Ø-AØ 84 17 3C Ø4 74 13 Ø2
337Ø  C1 3C Ø4 75 EB A2 84 17-32 CØ A2 85 17 E8 BB Ø3
338Ø  EB DE AØ 85 17 Ø2 C1 3C-Ø2 74 D5 73 Ø8 A2 85 17
339Ø  E8 A8 Ø3 EB CB BØ Ø3 A2-84 17 E8 9E Ø3 EB C1 BA
33AØ  97 3B E8 29 CF E8 93 Ø3-8B 1E 84 17 8B 16 38 33
33BØ  ØA CØ 1B DA 75 Ø3 E9 Ø1-Ø1 E9 DA ØØ 82 17 E8 79
33CØ  Ø1 E8 44 Ø3 53 69 7A 65-2Ø 63 68 61 6E 67 65 2Ø
33DØ  28 34 2D 36 29 ØØ E8 8D-Ø2 BØ 37 A2 85 3B BE 31

33EØ  39 89 36 89 3B BE 35 34-89 36 8C 3B AØ 83 17 DØ
33FØ  CØ DØ CØ DØ CØ E8 A8 Ø3-E8 F4 ØØ 72 1B AØ 83 17
34ØØ  Ø2 C1 FE C8 3C Ø6 73 FØ-FE CØ A2 83 17 DØ CØ DØ
341Ø  CØ DØ CØ E8 8A Ø3 EB EØ-BØ 3Ø A2 85 3B AØ 83 17
342Ø  DØ CØ DØ CØ DØ CØ E8 77-Ø3 BØ 37 A2 85 3B B8 35
343Ø  37 A3 8C 3B AØ 82 17 E8-66 Ø3 E8 B2 ØØ 72 15 AØ
344Ø  82 17 Ø2 C1 FE C8 3C 3Ø-73 FØ FE CØ A2 82 17 E8
345Ø  4E Ø3 EB E6 BØ 3Ø A2 85-3B AØ 82 17 E8 41 Ø3 8B
346Ø  1E BC 31 8B 1F 8B 16 36-33 ØA CØ 1B DA 75 Ø3 EB
347Ø  49 9Ø Ø3 DA 53 89 16 82-17 E8 5A EA 5B 89 1E 82

348Ø  17 BB ØØ ØØ 89 1E 89 17-89 1E 8B 17 89 1E 8D 17
349Ø  AØ 86 17 A2 8F 17 BØ 3Ø-A2 85 3B E8 C8 Ø1 E8 91

34AØ  Ø1 E8 Ø8 Ø4 E8 69 ED E8-F8 E9 E8 BØ F3 E8 B9 EA
34BØ  E8 77 F5 BA 81 3B E8 15-CE C3 BØ 3Ø A2 85 3B E9
34CØ  64 Ø1 BB 83 17 89 1E BC-31 AØ 83 17 DØ CØ DØ CØ
34DØ  DØ CØ 8A D8 AØ 82 17 8A-F8 BE 31 39 89 36 89 3B
34EØ  BE 35 34 89 36 8C 3B BE-35 37 E8 A8 Ø2 EB C4 E8
34FØ  46 F9 B1 FF 3C 34 75 Ø1-C3 B1 Ø1 3C 36 75 Ø1 C3
35ØØ  3C ØD F9 75 Ø6 BØ 3Ø A2-85 3B C3 3C 1B 75 EØ BØ
351Ø  3Ø A2 85 3B BB 36 33 BA-82 17 B9 Ø4 ØØ 87 F3 87

352Ø  FA 8C C8 8E CØ FC F3 A4-87 F3 87 FA E8 9A FF E8
353Ø  ØC Ø2 58 E9 F3 ØØ ØØ ØØ-ØØ ØØ BB 82 17 BA 36 33
354Ø  B9 Ø4 ØØ 87 F3 87 FA 8C-C8 8E CØ FC F3 A4 87 F3
355Ø  87 FA E9 Ø8 F3 B8 Ø7 ØC-CD 21 5Ø BA F6 16 E8 6D
356Ø  CD 58 C3 E8 9F Ø1 43 6F-6C 6F 72 28 34 2D 36 29
357Ø  ØØ E8 F2 ØØ AØ 88 17 BF-48 5A A2 BE 2E C7 Ø6 BF
358Ø  2E Ø4 ØØ E8 7D FA E8 CC-FF 3C 1B 74 Ø8 3C 2Ø 74
359Ø  Ø4 3C ØD 75 Ø6 E8 3E FA-E9 8B ØØ 3C 34 75 23 AØ
35AØ  88 17 FE C8 78 EØ A2 88-17 BF 48 5A A2 BE 2E C7
35BØ  Ø6 BF 2E Ø4 ØØ 5Ø E8 4A-FA 58 E8 B3 ØØ E8 F3 FE

35CØ  EB C4 3C 36 75 23 AØ 88-17 FE CØ 3C 1Ø 7F B7 A2
35DØ  88 17 BF 48 5A A2 BE 2E-C7 Ø6 BF 2E Ø4 ØØ 5Ø E8
35EØ  21 FA 58 E8 8A ØØ E8 CA-FE EB 9B 8Ø 3E 9E 17 Ø1
35FØ  75 Ø3 E9 6E FF E8 ØD Ø1-43 6F 6C 6F 72 28 3Ø 7E
36ØØ  38 29 3D ØØ E8 4E FF 3C-3Ø 72 1E 3C 39 73 1A 52
361Ø  5Ø 8A DØ B4 Ø2 CD 21 58-5A 24 ØF A2 88 17 5Ø E8
362Ø  44 ØØ 58 E8 4A ØØ E8 8A-FE E8 3A ØØ E8 FB F3 E8
363Ø  A7 F3 BA 63 39 E8 96 CC-C3 E8 21 F2 E8 C9 ØØ 53
364Ø  75 72 65 28 59 2F 65 6C-73 65 29 3D ØØ E8 E8 F7
365Ø  24 DF 3C 46 74 F7 3C 4D-74 F3 3C 59 75 ØF 52 8A

366Ø  DØ B4 Ø2 CD 21 5A BA F6-16 E8 62 CC C3 58 EB B9
367Ø  B4 Ø5 3C 1Ø 75 Ø3 B8 Ø6-Ø6 Ø4 Ø4 37 86 C4 ØD 3Ø
368Ø  3Ø A3 7C 3B BA 64 3B E8-44 CC C3 1B 5B 3Ø 31 3B
369Ø  3Ø 31 48 24 24 2E A1 9Ø-34 25 ØF ØF 86 C4 Ø4 Ø1
36AØ  37 ØD 3Ø 3Ø 86 C4 2E A3-9Ø 34 BA 8B 34 E8 1E CC
36BØ  C3 2E A1 9Ø 34 25 ØF ØF-86 C4 2C Ø1 3F ØD 3Ø 3Ø
36CØ  86 C4 2E A3 9Ø 34 BA 8B-34 E8 Ø2 CC C3 2E A1 8D
36DØ  34 25 ØF ØF 86 C4 Ø4 31-37 ØD 3Ø 3Ø 86 C4 2E A3
36EØ  8D 34 BA 8B 34 E8 E6 CB-C3 2E A1 8D 34 25 ØF ØF
36FØ  86 C4 2C 31 3F ØD 3Ø 3Ø-86 C4 2E A3 8D 34 BA 8B

37ØØ  34 E8 CA CB C3 E8 55 F1-BB 31 39 89 1E 8D 34 BB
371Ø  36 32 89 1E 9Ø 34 BA 8B-34 E8 B2 CB 8B EC 87 5E
372Ø  ØØ 8A Ø7 43 ØA CØ 74 ØA-52 8A DØ B4 Ø2 CD 21 5A
373Ø  EB EF 87 5E ØØ E8 85 ØØ-E9 78 FD E8 75 FD 8Ø 3E
374Ø  9E 17 Ø1 74 29 AØ 84 17-3C Ø3 75 Ø9 BA ØA 3B E8
375Ø  7C CB EB 4Ø 9Ø AØ 85 17-ØA CØ 75 Ø9 BA 19 3B E8
376Ø  6C CB EB 3Ø 9Ø BA 28 3B-E8 63 CB EB 27 9Ø AØ 84
377Ø  17 3C Ø3 75 Ø9 BA 37 3B-E8 53 CB EB 17 9Ø AØ 85
378Ø  17 ØA CØ 75 Ø9 BA 46 3B-E8 43 CB EB Ø7 9Ø BA 55
379Ø  3B E8 3A CB C3 8A C3 E8-Ø6 ØØ 89 36 8C 3B 8A C7

37AØ  B1 2Ø 2C ØA 72 Ø7 8Ø C9-1Ø FE C1 EB F5 Ø4 3A 88
37BØ  ØE 8F 3B A2 9Ø 3B BA 83-3B E8 12 CB C3 BA FØ 16
37CØ  E8 ØB CB C3 BB Ø1 ØØ 2E-8B ØE 89 17 FE C5 BA 8Ø
37DØ  Ø2 Ø3 DA FE CD 75 FA Ø3-D9 C3 E8 4E ØØ 8B ØE 9Ø
37EØ  17 EB Ø8 9Ø E8 4A ØØ 8B-ØE 89 17 BA Ø6 ØØ EB 18
37FØ  9Ø 8B ØE 9Ø 17 51 E8 81-ØØ EB Ø9 9Ø 8B ØE 89 17
38ØØ  51 E8 7C ØØ 59 BA 5Ø ØØ-FE C5 FE CD 74 Ø4 Ø3 DA
381Ø  EB F8 8A C1 2C Ø8 72 Ø3-43 EB F9 B1 8Ø Ø4 Ø8 75
382Ø  Ø1 C3 DØ C9 FE C8 75 Ø1-C3 EB F7 AØ 8F 17 EB Ø4
383Ø  9Ø AØ 86 17 8A DØ B9 2Ø-Ø1 BB 4Ø 9Ø FE CA 75 Ø1

384Ø  C3 Ø3 D9 EB F7 5Ø AØ 82-17 8A DØ 33 CØ 8A FØ 8A
385Ø  F8 8A D8 8B ØE 83 17 Ø3-C2 FE C9 75 FA 8B DØ Ø2
386Ø  2E 9E 17 Ø3 DA FE CD 75-FA BA EØ 3B 87 D3 58 8A
387Ø  E8 FE CD 75 Ø1 C3 Ø3 DA-EB F7 AØ 8F 17 EB Ø4 9Ø
388Ø  AØ 86 17 8A C8 BB 74 ØB-BA Ø7 ØØ B5 Ø4 FE C9 75
389Ø  Ø1 C3 Ø3 DA FE CD 75 F5-BA 94 1D Ø3 DA EB E9 BA
38AØ  C2 39 E8 29 CA C3 BA Ø5-3B E8 22 CA AØ 82 17 8A
38BØ  26 83 17 F6 E4 8A 1E 84-17 Ø2 1E 9E 17 B7 ØØ F7
38CØ  E3 8B DØ 33 CØ BE D2 36-E8 37 ØØ BA 8E 38 E8 FD
38DØ  C9 C3 97 38 BA 38 9E 38-C1 38 A5 38 C8 38 AC 38

38EØ  CF 38 DD 38 ØØ 39 E4 38-Ø7 39 EB 38 ØE 39 F2 38
38FØ  15 39 24 39 48 39 2B 39-4F 39 32 39 56 39 39 39
39ØØ  5D 39 B9 ØC ØØ 8B 3C 83-C6 Ø2 E8 ØF ØØ Ø3 C2 48
391Ø  8B 3C 83 C6 Ø2 E8 Ø4 ØØ-4Ø E2 EA C3 51 B1 Ø4 B7
392Ø  ØØ 8A DC E8 ØC ØØ B7 ØØ-8A D8 83 C7 Ø2 E8 Ø2 ØØ
393Ø  59 C3 D3 E3 8Ø FF ØA 7C-Ø3 8Ø C7 Ø7 8Ø C7 3Ø 88
394Ø  3D B7 ØØ D3 E3 8Ø FF ØA-7C Ø3 8Ø C7 Ø7 8Ø C7 3Ø
395Ø  88 7D Ø1 C3 E8 DE ØØ 8B-1E 82 17 DØ E7 DØ E7 DØ
396Ø  E7 AØ 89 17 Ø2 C5 FE C8-2A C7 72 12 FE CØ F6 D8
397Ø  8A F8 Ø2 C5 8A E8 AØ 92-17 Ø2 C7 A2 92 17 AØ 8A
```

```
398Ø   17 Ø2 C1 FE C8 2A C3 72-12 FE CØ F6 D8 8A D8 Ø2
399Ø   C1 8A C8 AØ 93 17 Ø2 C3-A2 93 17 E8 4C ØØ C6 45
39AØ   Ø1 ØØ C6 45 Ø2 ØØ BB 95-17 FE Ø7 AØ 9Ø 17 24 Ø7
39BØ   8A D8 AØ 89 17 24 Ø7 2A-C3 BF ØØ 85 BB 96 17 73
39CØ   Ø4 Ø4 Ø8 47 43 A2 BØ 24-89 1E 9B 1A ØA CØ 75 Ø1
39DØ   C3 5Ø AØ 95 17 8A E8 BB-96 17 DØ 1F 43 FE CD 75
39EØ   F9 58 FE C8 75 EB C3 E8-4B ØØ 8A C1 A2 94 17 AØ
39FØ   9Ø 17 24 Ø7 8A FØ BB 72-38 Ø2 C3 73 Ø2 FE C7 8A
3AØØ   D8 8A Ø7 BB 95 17 C6 Ø7-Ø1 BF 96 17 88 Ø5 8A C5
3A1Ø   Ø2 C6 2C Ø8 72 ØB 75 Ø1-C3 FE Ø7 47 C6 Ø5 FF EB

3A2Ø   F1 F6 D8 BB 79 38 Ø2 C3-73 Ø2 FE C7 8A D8 8A Ø7
3A3Ø   22 Ø5 88 Ø5 C3 8B 1E 8B-17 8B 16 8D 17 8A C6 2A
3A4Ø   C7 8A C8 73 ØA F6 D8 8A-C8 8A C6 8A F7 8A F8 8A
3A5Ø   C2 2A C3 8A E8 73 ØA F6-D8 8A E8 8A C2 8A D3 8A
3A6Ø   D8 89 1E 9Ø 17 89 16 92-17 FE C5 FE C1 C3 37 15
3A7Ø   26 Ø4 FF 7F 3F 1F ØF Ø7-Ø3 Ø1 FE FC F8 FØ EØ CØ
3A8Ø   8Ø 31 32 33 34 35 36 37-38 39 41 42 43 44 1B 5B
3A9Ø   31 3B 35 33 48 31 3A 2Ø-2Ø 2Ø 2Ø 2Ø 32 3A 2Ø 2Ø
3AAØ   2Ø 2Ø 2Ø 33 3A 2Ø 2Ø 2Ø-2Ø 2Ø 34 3A 2Ø 2Ø 2Ø 2Ø
3ABØ   2Ø 1B 5B 32 3B 35 33 48-2Ø 7E 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø

3ACØ   7E 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 7E-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 7E 2Ø
3ADØ   2Ø 2Ø 2Ø 2Ø 1B 5B 37 3B-35 33 48 35 3A 2Ø 2Ø 2Ø
3AEØ   2Ø 2Ø 36 3A 2Ø 2Ø 2Ø 2Ø-2Ø 37 3A 2Ø 2Ø 2Ø 2Ø 2Ø
3AFØ   38 3A 2Ø 2Ø 2Ø 2Ø 2Ø 1B-5B 38 3B 35 33 48 2Ø 7E
3BØØ   2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 7E 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 7E 2Ø 2Ø
3B1Ø   2Ø 2Ø 2Ø 2Ø 7E 2Ø 2Ø 2Ø-2Ø 2Ø 1B 5B 31 33 3B 35
3B2Ø   33 48 39 3A 2Ø 2Ø 2Ø 2Ø-2Ø 41 3A 2Ø 2Ø 2Ø 2Ø 2Ø
3B3Ø   42 3A 2Ø 2Ø 2Ø 2Ø 2Ø 43-3A 2Ø 2Ø 2Ø 2Ø 2Ø 1B 5B
3B4Ø   31 34 3B 35 33 48 2Ø 7E-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 7E 2Ø
3B5Ø   2Ø 2Ø 2Ø 2Ø 2Ø 7E 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 7E 2Ø 2Ø 2Ø

3B6Ø   2Ø 2Ø 24 1B 5B 31 39 3B-36 31 48 2Ø 2Ø 2Ø 2Ø 2Ø
3B7Ø   2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 24 Ø7
3B8Ø   1B 5B 31 39 3B 36 31 48-3Ø 2Ø 2Ø 1B 5B 31 39 3B
3B9Ø   36 35 48 42 1B 5B 31 39-3B 36 36 48 2Ø 1B 5B 31
3BAØ   39 3B 36 37 48 52 1B 5B-31 39 3B 36 38 48 2Ø 1B

3BBØ   5B 31 39 3B 36 39 48 47-1B 5B 31 39 3B 37 3Ø 48
3BCØ   2Ø 24 1B 5B 3E 35 68 1B-5B 31 38 3B 35 32 48 2Ø
3BDØ   2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 1B-29 33 98 95 95 95 95 95
3BEØ   95 95 95 95 95 95 95 95-95 95 95 95 95 95 99 1B
3BFØ   29 3Ø 1B 5B 31 39 3B 35-32 48 53 3D 2Ø 2Ø 78 2Ø

3CØØ   2Ø 2Ø 1B 29 33 96 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
3C1Ø   2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 96 1B 29 3Ø 1B 5B 32
3C2Ø   3Ø 3B 35 32 48 44 3D 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 1B 29 33
3C3Ø   9A 95 95 95 95 95 95 95-95 95 95 95 95 95 95 95
3C4Ø   95 95 95 95 9B 1B 29 3Ø-1B 5B 32 31 3B 35 32 48
3C5Ø   43 3D 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 4D 3A 6D
3C6Ø   6F 76 65 2Ø 2Ø 46 3A 66-2E 62 6F 78 2Ø 1B 5B 32
3C7Ø   32 3B 35 32 48 A5 3D 28-2Ø 3Ø 2C 2Ø 3Ø 29 2D 31
3C8Ø   2Ø 2Ø 3D 3A 7Ø 6F 7Ø 2Ø-2Ø 2Ø 2B 3A 7Ø 75 73 68
3C9Ø   2Ø 2Ø 2Ø 1B 5B 32 33 3B-35 32 48 5A 3D 28 2Ø 3Ø

3CAØ   2C 2Ø 3Ø 29 2D 31 2Ø 2Ø-54 3A 74 65 73 74 2Ø 2Ø
3CBØ   5Ø 3A 7Ø 61 6C 6C 65 74-2Ø 1B 5B 32 34 3B 35 32
3CCØ   48 58 3D 28 2Ø 3Ø 2C 2Ø-3Ø 29 2D 31 2Ø 2Ø 5E 52
3CDØ   2C 5E 41 3A 72 65 61 64-2Ø 64 61 74 61 2Ø 2Ø 1B
3CEØ   5B 32 35 3B 35 32 48 5E-58 2C 5E 59 3A CA DD C3
3CFØ   DD 2Ø 2Ø 2Ø 5E 44 2C 5E-46 3A 6D 61 6B 65 2Ø 64
3DØØ   61 74 61 2Ø 24 1B 5B 32-4A 24 1B 5B 32 3Ø 3B 35
3D1Ø   34 48 42 52 47 2Ø 2Ø 2Ø-24 1B 5B 32 3Ø 3B 35 34
3D2Ø   48 54 3Ø 42 52 47 2Ø 24-1B 5B 32 3Ø 3B 35 34 48
3D3Ø   54 31 42 52 47 2Ø 24 1B-5B 32 3Ø 3B 35 34 48 42

3D4Ø   52 47 4C 2Ø 2Ø 24 1B 5B-32 3Ø 3B 35 34 48 54 3Ø
3D5Ø   42 52 47 4C 24 1B 5B 32-3Ø 3B 35 34 48 54 31 42
3D6Ø   52 47 4C 24 1B 5B 32 31-3B 35 34 48 2Ø 2Ø 2Ø 2Ø
3D7Ø   2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 1B-5B 32 31 3B 35 42 48 DB
3D8Ø   24 Ø7 24 1B 5B 3Ø 6D 1B-5B 31 39 3B 35 34 48 34
3D9Ø   38 24 1B 5B 37 6D 24 1B-5B 3Ø 6D 24 1B 5B 3E 35
3DAØ   68 41 3A 24 1B 5B 3E 35-68 42 3A 24 1B 5B 3E 35
3DBØ   68 43 3A 24 1B 5B 3E 35-68 44 3A 24 1B 5B 73 1B
3DCØ   5B 3E 35 68 1B 5B 3Ø 4B-1B 5B 31 39 3B 38 3Ø 48
3DDØ   1B 29 33 96 1B 5B 75 1B-5B 3E 35 6C 1B 29 3Ø 24
```

リスト A-2 MAPEDIT

```
ØØØØ  4D 5A 35 Ø1 12 ØØ 18 ØØ-2Ø ØØ 33 2D FF FF E4 Ø9
ØØ1Ø  ØØ Ø2 4F B9 ØØ ØØ ØØ ØØ-1E ØØ ØØ ØØ Ø1 ØØ 6E ØØ
ØØ2Ø  ØØ ØØ E6 ØØ ØØ ØØ 38 Ø1-ØØ ØØ 88 Ø1 ØØ ØØ CA Ø1
ØØ3Ø  ØØ ØØ 1C Ø2 ØØ ØØ 6A Ø2-ØØ ØØ 92 Ø2 ØØ ØØ BB Ø2
ØØ4Ø  ØØ ØØ CB Ø2 ØØ ØØ AØ Ø4-ØØ ØØ CC Ø4 ØØ ØØ FC ØC
ØØ5Ø  ØØ ØØ 2B ØD ØØ ØØ BC 13-ØØ ØØ 86 14 ØØ ØØ F3 14
ØØ6Ø  ØØ ØØ Ø4 15 ØØ ØØ BD 16-ØØ ØØ F2 16 ØØ ØØ 72 17
ØØ7Ø  ØØ ØØ Ø1 18 ØØ ØØ AB 18-ØØ ØØ 54 1C ØØ ØØ ØØ ØØ
ØØ8Ø  ØØ ØD ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØ9Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ

ØØAØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØBØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØCØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØDØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØEØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
ØØFØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø1ØØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø11Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø12Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø13Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ

Ø14Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø15Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø16Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø17Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø18Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø19Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø1AØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø1BØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø1CØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø1DØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ

Ø1EØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø1FØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø2ØØ  B8 ØØ 4Ø CD 18 B8 ØØ 42-B9 ØØ CØ CD 18 FC 8C C8
Ø21Ø  8E D8 8E CØ Ø6 BA 37 12-B8 Ø6 35 CD 21 89 1E 38
Ø22Ø  12 8C Ø6 3A 12 Ø7 B8 Ø6-25 CD 21 BØ ØØ A2 62 1C
Ø23Ø  B4 19 CD 21 Ø4 41 A2 7F-1C A2 D2 1F A2 7F 1C A2
Ø24Ø  96 1C A2 AC 1C A2 1F 1E-A2 68 1E AØ 7F 1C A2 1F
Ø25Ø  1E E9 1Ø 1Ø 92 6E 8F E3-CA DF CØ BØ DD 2Ø DB BØ
Ø26Ø  C4 DE BF 73 1E BE 54 ØØ-B9 ØE ØØ F3 A4 B8 Ø4 ØA
Ø27Ø  A3 54 1C A1 31 21 BF E7-1C E8 2Ø ØØ A1 58 1C 83

Ø28Ø  3E 58 1C ØØ 74 15 A3 C6-ØØ BB 8Ø Ø1 9Ø BA ØØ ØØ
Ø29Ø  F7 F3 A3 ØE 1C C7 Ø6 1A-1C ØØ ØØ C3 A3 33 ØE C7
Ø2AØ  Ø6 6D 1C ØØ ØØ C7 Ø6 58-1C ØØ ØØ C6 Ø6 C5 ØØ ØØ
Ø2BØ  57 E8 34 Ø5 5F 8Ø 3E C5-ØØ Ø1 75 Ø8 B9 11 ØØ BE
Ø2CØ  C1 1C F3 A4 C3 ØØ ØØ ØØ-ØØ ØØ ØØ ØØ 92 6E 8F E3
Ø2DØ  CA DF CØ BØ DD 2Ø 92 C7-89 C1 BF 73 1E BE CC ØØ
Ø2EØ  B9 ØE ØØ F3 A4 B8 Ø4 ØA-A3 54 1C A1 31 21 A3 33
Ø2FØ  ØE A1 C6 ØØ A3 6D 1C C7-Ø6 58 1C ØØ ØØ E8 E8 Ø4
Ø3ØØ  A1 58 1C 83 3E 58 1C ØØ-74 13 Ø1 Ø6 C6 ØØ A1 C6
Ø31Ø  ØØ BB 8Ø Ø1 9Ø BA ØØ ØØ-F7 F3 A3 ØE 1C C3 92 6E

Ø32Ø  8F E3 CA DF CØ BØ DD 2Ø-BE BØ CC DE BF B1 1E BE
Ø33Ø  1E Ø1 B9 ØE ØØ F3 A4 B8-Ø4 ØA A3 54 1C A1 C6 ØØ
Ø34Ø  A3 C8 ØØ 23 CØ 74 24 B9-11 ØØ BE E7 1C BF C1 1C
Ø35Ø  F3 A4 B9 11 ØØ BE C1 1C-BF D4 1C F3 A4 E8 A3 Ø7
Ø36Ø  B9 11 ØØ BE D4 1C BF E7-1C F3 A4 C3 ØØ ØØ 2Ø 93
Ø37Ø  47 2Ø CA DF CØ BØ DD 2Ø-DB BØ C4 DE BF 73 1E BE
Ø38Ø  6E Ø1 B9 ØE ØØ F3 A4 B8-64 13 A3 54 1C A1 2F 21
Ø39Ø  BF FA 1C E8 Ø6 FF A1 58-1C 83 3E 58 1C ØØ 74 ØF
Ø3AØ  A3 6C Ø1 BB ØØ Ø2 9Ø BA-ØØ ØØ F7 F3 A3 ØA 1C C3
Ø3BØ  2Ø 93 47 2Ø CA DF CØ BØ-DD 2Ø 92 C7 89 C1 BF 73

Ø3CØ  1E BE BØ Ø1 B9 ØE ØØ F3-A4 B8 64 13 A3 54 1C A1
Ø3DØ  2F 21 A3 33 ØE A1 6C Ø1-A3 6D 1C C7 Ø6 58 1C ØØ
Ø3EØ  ØØ E8 Ø4 Ø4 A1 58 1C 83-3E 58 1C ØØ 74 13 Ø1 Ø6
Ø3FØ  6C Ø1 A1 6C Ø1 BB ØØ Ø2-9Ø BA ØØ ØØ F7 F3 A3 ØA
Ø4ØØ  1C C3 2Ø 93 47 2Ø CA DF-CØ BØ DD 2Ø BE BØ CC DE
Ø41Ø  BF B1 1E BE Ø2 Ø2 B9 ØE-ØØ F3 A4 B8 64 13 A3 54
Ø42Ø  1C A1 6C Ø1 A3 C8 ØØ 23-CØ 74 24 B9 11 ØØ BE FA
Ø43Ø  1C BF C1 1C F3 A4 B9 11-ØØ BE C1 1C BF D4 1C F3
Ø44Ø  A4 E8 BF Ø6 B9 11 ØØ BE-D4 1C BF FA 1C F3 A4 C3
Ø45Ø  92 6E 8F E3 2Ø C3 DE BØ-CØ 2Ø DB BØ C4 DE BF 73

Ø46Ø  1E BE 5Ø Ø2 B9 ØE ØØ F3-A4 B8 E4 1F A3 54 1C A1
Ø47Ø  33 21 BF ØD 1D E8 24 FE-A1 58 1C 83 3E 58 1C ØØ
Ø48Ø  74 1E 1E Ø6 D1 E8 8B FØ-8B C8 48 A3 1A 1C A3 2A
Ø49Ø  1C B8 E4 1F 8E D8 8E CØ-BF ØØ 8Ø FC F3 A4 Ø7 1F
Ø4AØ  C3 92 6E 8F E3 2Ø C3 DE-BØ CØ 2Ø BE BØ CC DE BF
Ø4BØ  B1 1E BE A1 Ø2 B9 ØE ØØ-F3 A4 B8 E4 1F A3 54 1C
Ø4CØ  8B 1E 2A 1C 23 DB 74 49-1E Ø6 B8 E4 1F 8E D8 8E
Ø4DØ  CØ 8Ø 3F FF 74 Ø3 C6 Ø7-FF 43 8B CB 8B FB D1 E3
Ø4EØ  2E 89 1E C8 ØØ BE ØØ 8Ø-FC F3 A4 Ø7 1F B9 11 ØØ
Ø4FØ  BE ØD 1D BF C1 1C F3 A4-B9 11 ØØ BE C1 1C BF D4

Ø5ØØ  1C F3 A4 E8 FD Ø5 B9 11-ØØ BE D4 1C BF ØD 1D F3
Ø51Ø  A4 C3 B2 41 EB 1Ø 9Ø B2-42 EB ØB 9Ø B2 43 EB Ø6
Ø52Ø  9Ø B2 44 EB Ø1 9Ø 88 16-5F Ø3 52 BA 5F Ø3 B4 3B
Ø53Ø  CD 21 5A 72 26 88 16 D2-1F 88 16 7F 1C 88 16 96
Ø54Ø  1C 88 16 AC 1C 88 16 1F-1E 88 16 68 1E 8Ø EA 41
Ø55Ø  B4 ØE CD 21 BA CD 1F B4-Ø9 CD 21 E8 FE ØØ C3 42
Ø56Ø  3A 5C ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø57Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø58Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
Ø59Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ

Ø5AØ  ØØ ØØ BA EØ 1D B4 Ø9 CD-21 BØ ØØ A2 62 1C C3 E8
Ø5BØ  8B ØA 75 Ø3 E9 A1 ØØ BA-3Ø 1E B4 Ø9 CD 21 B8 ØØ
Ø5CØ  ØØ A3 5C 1C BA EB 2Ø B4-1A CD 21 BF AC 1C AØ 7F
Ø5DØ  1C 88 Ø5 8B D7 B9 ØØ ØØ-B4 4E CD 21 73 Ø3 EB 78
Ø5EØ  9Ø 3D ØC ØØ 75 Ø3 EB 7Ø-9Ø E8 7Ø ØØ 51 52 56 BE
Ø5FØ  EB 2Ø 83 C6 1E B9 ØC ØØ-8A 14 B4 Ø6 CD 21 46 E2
Ø6ØØ  F7 B2 2Ø B4 Ø6 CD 21 5E-5A 59 BF EB 2Ø 83 C7 1E
Ø61Ø  B8 2Ø ØØ B9 Ø6 ØØ F3 AB-B4 4F CD 21 72 3A 3D ØC
Ø62Ø  ØØ 74 35 A1 5C 1C 4Ø A3-5C 1C 3D Ø4 ØØ 7C Ø9 B8
Ø63Ø  ØØ ØØ A3 5C 1C E8 24 ØØ-51 52 56 BE EB 2Ø 83 C6

Ø64Ø  1E B9 ØC ØØ 8A 14 B4 Ø6-CD 21 46 E2 F7 B2 2Ø B4
Ø65Ø  Ø6 CD 21 5E 5A 59 EB B2-E8 Ø1 ØØ C3 AØ 62 1C FE
Ø66Ø  CØ 3A Ø6 14 1E 7C ØA BA-FØ 1D B4 Ø9 CD 21 EB ØB
Ø67Ø  9Ø A2 62 1C BA 7F 1D B4-Ø9 CD 21 C3 A1 65 1C 4Ø
Ø68Ø  3B Ø6 77 1C 7E ØD C7 Ø6-6B 1C Ø1 ØØ B8 ØØ ØØ A3
Ø69Ø  65 1C C3 A3 65 1C BA 7F-1D B4 Ø9 CD 21 C3 Ø6 B8
Ø6AØ  14 Ø2 8E CØ BE EB 2Ø 83-C6 1E 2E 8B 3E 67 1C 2E
Ø6BØ  89 3E 69 1C B9 Ø8 ØØ F3-A5 2E 3B 3E 6F 1C 7C Ø3
Ø6CØ  BF ØØ ØØ 2E 89 3E 67 1C-Ø7 C3 Ø6 B8 14 Ø2 8E CØ
Ø6DØ  33 CØ 8B F8 A3 6B 1C B9-ØØ 2Ø F3 AB Ø7 BA 49 1E

Ø6EØ  B4 Ø9 CD 21 B8 ØØ ØØ A3-5C 1C A3 67 1C A3 63 1C
Ø6FØ  A3 65 1C A3 75 1C B4 31-88 26 B4 2Ø BØ 3Ø A2 B3
Ø7ØØ  2Ø A2 BØ 2Ø BØ 34 A2 B1-2Ø BA EB 2Ø B4 1A CD 21
Ø71Ø  BF 7F 1C AØ 7F 1C 88 Ø5-8B D7 B9 ØØ ØØ B4 4E CD
Ø72Ø  21 73 Ø3 E9 89 ØØ BA 49-1E B4 Ø9 CD 21 E8 4C FF
Ø73Ø  51 52 56 BE EB 2Ø 83 C6-1E B9 ØC ØØ 8A 14 B4 Ø6
Ø74Ø  CD 21 46 E2 F7 B2 2Ø B4-Ø6 CD 21 5E 5A 59 E8 4D
Ø75Ø  FF BF EB 2Ø 83 C7 1E B8-2Ø ØØ B9 Ø6 ØØ F3 AB B4
Ø76Ø  4F CD 21 73 Ø3 EB 7A 9Ø-A1 5C 1C 4Ø A3 5C 1C 3B
Ø77Ø  Ø6 79 1C 7C 1Ø B8 ØØ ØØ-A3 5C 1C 83 3E 6B 1C ØØ

Ø78Ø  75 28 E8 F7 FE 83 3E 6B-1C ØØ 75 1E 51 52 56 BE
Ø79Ø  EB 2Ø 83 C6 1E B9 ØC ØØ-8A 14 B4 Ø6 CD 21 46 E2
Ø7AØ  F7 B2 2Ø B4 Ø6 CD 21 5E-5A 59 E8 F1 FE EB A2 BA
Ø7BØ  36 2Ø B4 Ø9 CD 21 E8 8Ø-Ø1 BF 81 1C B9 ØC ØØ BØ
Ø7CØ  2Ø F3 AA BF 81 1C C6 Ø5-2A C6 45 Ø1 2E C6 45 Ø2
Ø7DØ  2A B9 1Ø ØØ BØ 2Ø BF C1-1C 88 Ø5 47 E2 FB EB Ø7
Ø7EØ  9Ø C7 Ø6 7B 1C Ø1 ØØ C3-E8 52 Ø8 75 Ø3 E9 68 FE
Ø7FØ  B8 ØØ ØØ A3 7B 1C C6 Ø6-C5 ØØ ØØ E8 CC FE 83 3E
Ø8ØØ  7B 1C ØØ 75 ØD BØ Ø1 A2-62 1C C6 Ø6 C5 ØØ ØØ E9
Ø81Ø  26 Ø1 B8 ØØ ØØ A3 3B ØE-A3 5E 1C A3 6Ø 1C A3 65

Ø82Ø  1C A3 63 1C E8 AE Ø6 B8-ØØ ØC CD 21 B8 ØØ ØØ CD
Ø83Ø  18 BF 3C 1C 8B ØE 39 ØE-F2 AF 74 Ø5 E8 2C Ø1 EB
Ø84Ø  E6 5Ø BA A8 2Ø B4 Ø9 CD-21 58 8Ø 3E 33 1D ØØ 75
Ø85Ø  Ø3 E9 DØ ØØ C6 Ø6 33 1D-ØØ 9Ø 3C ØD 74 Ø3 E9 C3
Ø86Ø  ØØ BF C1 1C 8A 26 7F 1C-B9 1Ø ØØ 8A Ø5 3C 3A 74
Ø87Ø  13 3C 2Ø 74 Ø6 3C 5C 74-Ø2 8A EØ 47 E2 ED BF C1
Ø88Ø  1C EB 67 9Ø 47 8Ø FC 41-7C 6Ø 8Ø FC 44 7F 22 88
Ø89Ø  26 5F Ø3 5Ø BA 5F Ø3 B4-3B CD 21 58 72 4C 88 26
Ø8AØ  7F 1C 88 26 AC 1C 88 26-1F 1E 88 26 68 1E EB 31
Ø8BØ  9Ø 8Ø FC 61 7C 34 8Ø FC-64 7F 2F 8Ø EC 2Ø 88 26

Ø8CØ  5F Ø3 5Ø BA 5F Ø3 B4 3B-CD 21 58 72 1D 88 26 7F
Ø8DØ  1C 88 26 96 1C 88 26 AC-1C 88 26 1F 1E 88 26 68
Ø8EØ  1E 8Ø EC 41 8A D4 B4 ØE-CD 21 AØ 7F 1C A2 96 1C
Ø8FØ  BE 98 1C B9 ØD ØØ 87 F7-F3 A4 BØ 2A BF C1 1C B9
Ø9ØØ  1Ø ØØ F2 AE 74 ØC BØ 3F-BF C1 1C B9 1Ø ØØ F2 AE
Ø91Ø  75 ØE BE 98 1C BF 81 1C-B9 ØD ØØ F3 A4 E9 C8 FE
```

```
Ø92Ø  E8 3B Ø6 C3 D1 E1 BB 48-1C Ø3 D9 FF 17 2E 83 3E
Ø93Ø  3B ØE ØØ 75 Ø3 E9 EF FE-C3 BF 81 1C B9 ØD ØØ B2
Ø94Ø  5B B4 Ø6 CD 21 B2 2Ø B4-Ø6 CD 21 8A 15 8Ø FA 2Ø
Ø95Ø  74 ØC 8Ø FA ØØ 74 Ø7 B4-Ø6 CD 21 47 EØ ED B2 2Ø

Ø96Ø  B4 Ø6 CD 21 B2 5D B4 Ø6-CD 21 C3 8Ø 3E 33 1D ØØ
Ø97Ø  75 4Ø 5Ø BØ 2Ø B9 1Ø ØØ-BF C1 1C F3 AA C7 Ø6 1B
Ø98Ø  1F ØØ ØØ C7 Ø6 19 1F ØØ-ØØ C7 Ø6 22 1F 32 32 BA
Ø99Ø  17 1F B4 Ø9 CD 21 BA 1D-1F B4 Ø9 CD 21 BA 26 1F
Ø9AØ  B4 Ø9 CD 21 BA A2 2Ø B4-Ø9 CD 21 C6 Ø6 33 1D Ø1
Ø9BØ  9Ø 58 3C Ø8 74 66 3D ØØ-38 75 Ø3 E9 8C ØØ 3D ØØ
Ø9CØ  39 75 Ø3 E9 BB ØØ A8 8Ø-75 Ø7 3C 2Ø 73 Ø3 E9 2A
Ø9DØ  Ø1 8B DØ BF C1 1C A1 1B-1F Ø3 F8 4Ø 3D 1Ø ØØ 7C
Ø9EØ  ØA BA 17 1F B4 Ø9 CD 21-E9 1Ø Ø1 3B Ø6 19 1F 7C
Ø9FØ  Ø3 A3 19 1F A3 1B 1F 88-15 B4 Ø2 CD 21 2E A1 22

ØAØØ  1F 86 C4 25 ØF ØF Ø4 Ø1-37 ØD 3Ø 3Ø 86 C4 2E A3
ØA1Ø  22 1F BA 1D 1F B4 Ø9 CD-21 E9 DF ØØ A1 1B 1F 48
ØA2Ø  3D ØØ ØØ 7D Ø3 E9 D3 ØØ-A3 1B 1F 2E A1 22 1F 86
ØA3Ø  C4 25 ØF ØF 2C Ø1 3F ØD-3Ø 3Ø 86 C4 2E A3 22 1F
ØA4Ø  BA 1D 1F B4 Ø9 CD 21 E9-B1 ØØ A1 1B 1F 4Ø 3D 1Ø
ØA5Ø  ØØ 7C Ø3 E9 A5 ØØ 3B Ø6-19 1F 7E Ø3 E9 9C ØØ A3
ØA6Ø  1B 1F 2E A1 22 1F 86 C4-25 ØF ØF Ø4 Ø1 37 ØD 3Ø
ØA7Ø  3Ø 86 C4 2E A3 22 1F BA-1D 1F B4 Ø9 CD 21 EB 7B
ØA8Ø  9Ø BA A8 2Ø B4 Ø9 CD 21-BA 1D 1F B4 Ø9 CD 21 BA
ØA9Ø  26 1F B4 Ø9 CD 21 A1 19-1F 48 3D ØØ ØØ 7C 5C 3B

ØAAØ  Ø6 1B 1F 7C Ø6 A3 19 1F-EB 1C 9Ø A3 19 1F A3 1B
ØABØ  1F 2E A1 22 1F 86 C4 25-ØF ØF 2C Ø1 3F ØD 3Ø 3Ø
ØACØ  86 C4 2E A3 22 1F B9 1Ø-ØØ BF C1 1C A1 1B 1F Ø3
ØADØ  F8 2B C8 8A 45 Ø1 88 Ø5-47 E2 F8 C6 Ø5 2Ø BA 2A
ØAEØ  1F B4 Ø9 CD 21 BF C1 1C-B9 1Ø ØØ 8A 15 B4 Ø2 CD
ØAFØ  21 47 E2 F7 BA 1D 1F B4-Ø9 CD 21 BA A2 2Ø B4 Ø9
ØBØØ  CD 21 C3 E8 37 Ø5 75 Ø3-E9 4D FB BA A8 2Ø B4 Ø9
ØB1Ø  CD 21 B8 33 3Ø 86 C4 A3-98 1E A3 1F 1F A3 2C 1F
ØB2Ø  BA 91 1E B4 Ø9 CD 21 BF-C1 1C C7 Ø6 1B 1F ØØ ØØ
ØB3Ø  C7 Ø6 19 1F ØØ ØØ C7 Ø6-22 1F 32 32 B9 1Ø ØØ 8A

ØB4Ø  15 8Ø FA 2Ø 74 29 8Ø FA-ØØ 74 24 B4 Ø2 CD 21 FF
ØB5Ø  Ø6 19 1F FF Ø6 1B 1F 2E-A1 22 1F 86 C4 25 ØF ØF
ØB6Ø  Ø4 Ø1 37 ØD 3Ø 3Ø 86 C4-2E A3 22 1F 47 E2 DØ C6
ØB7Ø  Ø6 33 1D Ø1 9Ø B8 ØØ ØC-CD 21 B4 ØØ CD 18 3C ØD
ØB8Ø  74 1F 3C 1B 75 Ø3 E9 43-Ø1 8Ø FC 3C 75 Ø6 B8 ØØ
ØB9Ø  38 EB Ø9 9Ø 8Ø FC 3B 75-Ø3 B8 Ø8 ØØ E8 CC FD EB
ØBAØ  D4 BF C1 1C 8A 26 7F 1C-B9 1Ø ØØ 8A Ø5 3C 3A 74
ØBBØ  13 3C 2Ø 74 Ø6 3C 5C 74-Ø2 8A EØ 47 E2 ED BE C1
ØBCØ  1C EB 5Ø 9Ø 8B F7 46 8Ø-FC 41 7D Ø3 E9 FD ØØ 8Ø
ØBDØ  FC 44 7E 13 8Ø FC 61 7D-Ø3 E9 FØ ØØ 8Ø FC 64 7E

ØBEØ  Ø3 E9 E8 ØØ 8Ø EC 2Ø 88-26 5F Ø3 5Ø BA 5F Ø3 B4
ØBFØ  3B CD 21 58 72 1D 88 26-7F 1C 88 26 96 1C 88 26
ØCØØ  AC 1C 88 26 1F 1E 88 26-68 1E 8Ø EC 41 8A D4 B4
ØC1Ø  ØE CD 21 BF 98 1C B9 ØD-ØØ F3 A4 BA EB 2Ø B4 1A
ØC2Ø  CD 21 BF 96 1C AØ 7F 1C-88 Ø5 8B D7 B9 ØØ ØØ B4
ØC3Ø  4E CD 21 72 1E E8 24 FA-BA 4D 1F B4 Ø9 CD 21 B8
ØC4Ø  Ø7 ØC CD 21 3C 59 74 ØB-3C 79 74 Ø7 3C DD 74 Ø3
ØC5Ø  EB 7A 9Ø B2 ØØ B4 36 CD-21 F7 E1 F7 E3 23 D2 75
ØC6Ø  14 2E 2B Ø6 C8 ØØ 73 ØD-E8 F1 F9 BA B5 1F B4 Ø9
ØC7Ø  CD 21 EB 58 9Ø B4 3C BA-96 1C B9 ØØ ØØ CD 21 72

ØC8Ø  41 A3 5A 1C 8B D8 1E BA-ØØ ØØ 8B ØE C8 ØØ A1 54
ØC9Ø  1C 8E D8 B4 4Ø CD 21 1F-9C B4 3E 8B 1E 5A 1C CD
ØCAØ  21 9D 72 1E 4Ø 48 75 ØD-E8 B1 F9 BA B5 1F B4 Ø9
ØCBØ  CD 21 EB 18 9Ø E8 A4 F9-BA 78 1F B4 Ø9 CD 21 EB
ØCCØ  ØB 9Ø E8 97 F9 BA A6 1F-B4 Ø9 CD 21 C6 Ø6 33 1D
ØCDØ  ØØ 9Ø B8 3Ø 33 A3 1F 1F-A3 2C 1F BA A8 2Ø B4 Ø9
ØCEØ  CD 21 E8 77 F9 C3 A1 63-1C 4Ø 3B Ø6 79 1C 7C Ø6
ØCFØ  E8 E2 Ø1 EB 34 9Ø 2E 8B-1E 5E 1C 83 C3 1Ø 2E 3B
ØDØØ  1E 67 1C 7D 24 A3 63 1C-BA AE 2Ø B4 Ø9 CD 21 E8
ØD1Ø  ØE Ø2 2E 3B 1E 71 1C 7C-Ø5 2E 2B 1E 71 1C 2E 89

ØD2Ø  1E 5E 1C E8 Ø4 ØØ E8 AC-Ø1 C3 2E A1 B3 2Ø 25 ØF
ØD3Ø  ØF 86 C4 Ø4 Ø3 37 ØD 3Ø-3Ø FE C4 86 C4 2E A3 B3
ØD4Ø  2Ø C3 A1 63 1C 48 3D ØØ-ØØ 7D Ø6 E8 87 Ø1 EB 3Ø
ØD5Ø  9Ø 2E 8B 1E 5E 1C 83 EB-1Ø 83 FB ØØ 7C 22 A3 63
ØD6Ø  1C BA AE 2Ø B4 Ø9 CD 21-E8 B5 Ø1 83 FB ØØ 7D Ø5
ØD7Ø  2E Ø3 1E 71 1C 2E 89 1E-5E 1C E8 Ø4 ØØ E8 55 Ø1
ØD8Ø  C3 2E A1 B3 2Ø 25 ØF ØF-86 C4 2C Ø3 3F ØD 3Ø 3Ø
ØD9Ø  FE CC 86 C4 2E A3 B3 2Ø-C3 A1 65 1C 48 3D ØØ ØØ
ØDAØ  7D 2E 2E A1 5E 1C 2D 4Ø-ØØ 3D ØØ ØØ 7D Ø3 EB 61
ØDBØ  9Ø 5Ø E8 6B Ø1 BA 8C 2Ø-B4 Ø9 CD 21 58 3D ØØ ØØ

ØDCØ  7D Ø5 2E Ø3 Ø6 71 1C E8-48 ØØ E8 Ø8 Ø1 EB 42 9Ø
ØDDØ  2E 8B 1E 5E 1C 83 EB 4Ø-83 FB ØØ 7D Ø3 EB 32 9Ø
ØDEØ  A3 65 1C BA AE 2Ø B4 Ø9-CD 21 E8 33 Ø1 83 FB ØØ
ØDFØ  7D Ø5 2E Ø3 1E 71 1C 2E-89 1E 5E 1C 2E A1 BØ 2Ø
ØEØØ  86 C4 2C 31 3F ØD 3Ø 3Ø-86 C4 2E A3 BØ 2Ø E8 C4
ØE1Ø  ØØ C3 5Ø 2E FF 36 B3 2Ø-8B D8 2E 8B ØE 63 1C 41
ØE2Ø  49 74 Ø8 83 EB 1Ø E8 58-FF E2 F8 B9 ØØ ØØ 2E 89
ØE3Ø  1E 5E 1C E8 EA ØØ E8 F1-FE 83 C3 1Ø 2E 3B 1E 67
ØE4Ø  1C 7D Ø8 41 2E 3B ØE 79-1C 7C E3 2E 8F Ø6 B3 2Ø
ØE5Ø  58 2E A3 5E 1C C3 A1 65-1C 4Ø 3B Ø6 77 1C 7C 32

ØE6Ø  2E A1 5E 1C Ø5 4Ø ØØ 2E-3B Ø6 67 1C 7C Ø3 EB 64
ØE7Ø  9Ø 5Ø E8 AB ØØ BA 7A 2Ø-B4 Ø9 CD 21 58 2E 3B Ø6
ØE8Ø  71 1C 7C Ø5 2E 2B Ø6 71-1C E8 86 FF E8 46 ØØ EB
ØE9Ø  43 9Ø 2E 8B 1E 5E 1C 83-C3 4Ø 2E 3B 1E 67 1C 7D
ØEAØ  33 A3 65 1C BA AE 2Ø B4-Ø9 CD 21 E8 72 ØØ 2E 3B
ØEBØ  1E 71 1C 7C Ø5 2E 2B 1E-71 1C 2E 89 1E 5E 1C 2E
ØECØ  A1 BØ 2Ø 86 C4 Ø4 31 37-ØD 3Ø 3Ø 86 C4 2E A3 BØ
ØEDØ  2Ø E8 Ø1 ØØ C3 BA D9 2Ø-B4 Ø9 CD 21 BA AE 2Ø B4
ØEEØ  Ø9 CD 21 E8 12 ØØ BA 59-1E B4 Ø9 CD 21 E8 Ø8 ØØ
ØEFØ  BA E2 2Ø B4 Ø9 CD 21 C3-53 51 1E B8 14 Ø2 8E D8

ØFØØ  2E 8B 1E 5E 1C BE C1 1C-B9 ØC ØØ 8A 17 2E 88 14
ØF1Ø  B4 Ø2 CD 21 43 46 E2 F3-2E C6 Ø4 ØØ 1F 59 5B C3
ØF2Ø  BA AE 2Ø B4 Ø9 CD 21 53-51 1E B8 14 Ø2 8E D8 2E
ØF3Ø  8B 1E 5E 1C B9 ØC ØØ 8A-17 B4 Ø2 CD 21 43 E2 F7
ØF4Ø  1F 59 5B BA E2 2Ø B4 Ø9-CD 21 C3 AØ 7F 1C A2 96
ØF5Ø  1C BE 98 1C BF C1 1C B9-ØD ØØ 87 F7 F3 A4 BA 96
ØF6Ø  1C BØ ØØ B4 3D CD 21 73-Ø3 E9 A4 ØØ 9Ø A3 5A 1C
ØF7Ø  8B D8 B8 Ø2 42 B9 ØØ ØØ-8B D1 CD 21 89 16 37 ØE
ØF8Ø  A3 35 ØE 23 D2 74 27 B4-3E 8B 1E 5A 1C CD 21 BA
ØF9Ø  49 1E B4 Ø9 CD 21 BA 3F-1F B4 Ø9 CD 21 B8 Ø1 ØØ

ØFAØ  A3 3B ØE BØ Ø1 A2 62 1C-BØ ØØ A2 C5 ØØ C3 B4 3E
ØFBØ  8B 1E 5A 1C CD 21 BA 96-1C BØ ØØ B4 3D CD 21 A3
ØFCØ  5A 1C 8B ØE 35 ØE 8B 16-6D 1C 8B C1 Ø3 C2 3B Ø6
ØFDØ  33 ØE 73 B3 89 ØE 58 1C-8B 1E 5A 1C A1 54 1C 1E
ØFEØ  8E D8 B4 3F CD 21 1F 9C-B4 3E 8B 1E 5A 1C CD 21
ØFFØ  9D 72 1D BA 49 1E B4 Ø9-CD 21 BA 34 1F B4 Ø9 CD
1ØØØ  21 B8 Ø1 ØØ A3 3B ØE BØ-Ø1 A2 62 1C A2 C5 ØØ C3
1Ø1Ø  BA 49 1E B4 Ø9 CD 21 B8-Ø1 ØØ A3 3B ØE BØ ØØ A2
1Ø2Ø  62 1C A2 C5 ØØ B9 1Ø ØØ-BØ 2Ø BF C1 1C 88 Ø5 47
1Ø3Ø  E2 FB C3 ØØ ØØ ØØ ØØ ØØ-ØØ Ø6 ØØ ØØ ØØ B4 Ø4 AØ

1Ø4Ø  7F 1C 3C 41 74 Ø4 3C 42-75 15 24 ØF Ø4 8F CD 1B
1Ø5Ø  8Ø E4 FØ 8Ø FC 6Ø 75 Ø7-BA 58 2Ø B4 Ø9 CD 21 C3
1Ø6Ø  3C 32 75 13 AØ BF ØE A2-CØ ØE FE CØ 3A Ø6 BE ØE
1Ø7Ø  7C 18 BØ ØØ EB 14 9Ø 3C-38 75 15 AØ BF ØE A2 CØ
1Ø8Ø  ØE FE C8 79 Ø5 AØ BE ØE-FE C8 A2 BF ØE E8 Ø1 ØØ
1Ø9Ø  C3 AØ CØ ØE B4 ØØ 8B F8-D1 E7 81 C7 C1 ØE FF 15
1ØAØ  BA E2 2Ø B4 Ø9 CD 21 AØ-BF ØE B4 ØØ 8B F8 D1 E7
1ØBØ  81 C7 C1 ØE FF 15 BA AB-1A B4 Ø9 CD 21 C3 Ø9 ØØ
1ØCØ  ØØ D3 ØE DB ØE E3 ØE EB-ØE F3 ØE FB ØE Ø3 ØF ØB
1ØDØ  ØF 13 ØF BA EB 19 B4 Ø9-CD 21 C3 BA ØØ 1A B4 Ø9

1ØEØ  CD 21 C3 BA 15 1A B4 Ø9-CD 21 C3 BA 2A 1A B4 Ø9
1ØFØ  CD 21 C3 BA 3F 1A B4 Ø9-CD 21 C3 BA 54 1A B4 Ø9
11ØØ  CD 21 C3 BA 69 1A B4 Ø9-CD 21 C3 BA 7F 1A B4 Ø9
111Ø  CD 21 C3 BA 95 1A B4 Ø9-CD 21 C3 5Ø AØ 3E 12 22
112Ø  CØ 75 Ø3 E9 3C Ø1 9Ø BA-FA 1B B4 Ø9 CD 21 C6 Ø6
113Ø  3E 12 ØØ 9Ø 8Ø 3E BF ØE-Ø1 75 1A E8 ED Ø4 E8 21
114Ø  F1 E8 5D Ø7 E8 ØC Ø3 E8-9Ø Ø3 E8 25 Ø9 C6 Ø6 BF
115Ø  ØE ØØ E9 ØD Ø1 8Ø 3E BF-ØE Ø2 75 21 83 3E ØE 1C
116Ø  6Ø 7D 1A E8 C5 Ø4 E8 71-F1 E8 35 Ø7 E8 E4 Ø2 E8
117Ø  68 Ø3 E8 FD Ø8 C6 Ø6 BF-ØE ØØ E9 E5 ØØ 8Ø 3E BF

118Ø  ØE Ø3 75 24 E8 A4 Ø4 E8-A2 F1 E8 BA Ø2 E8 B7 Ø2
119Ø  E8 B4 Ø2 BA 42 19 B4 Ø9-CD 21 E8 B6 Ø2 E8 3A Ø3
11AØ  C6 Ø6 BF ØE ØØ E9 BA ØØ-8Ø 3E BF ØE Ø4 75 1A E8
11BØ  79 Ø4 E8 A9 F2 E8 E9 Ø6-E8 98 Ø2 E8 1C Ø3 E8 B1
11CØ  Ø8 C6 Ø6 BF ØE ØØ E9 99-ØØ 8Ø 3E BF ØE Ø5 75 24
11DØ  E8 58 Ø4 E8 D9 F2 E8 6E-Ø2 E8 6B Ø2 E8 68 Ø2 BA
11EØ  42 19 B4 Ø9 CD 21 E8 6A-Ø2 E8 EE Ø2 C6 Ø6 BF ØE
11FØ  ØØ EB 6F 9Ø 8Ø 3E BF ØE-Ø6 75 17 E8 2D Ø4 E8 7B
12ØØ  F1 E8 9D Ø6 E8 4C Ø2 E8-DØ Ø2 C6 Ø6 BF ØE ØØ EB
121Ø  51 9Ø 8Ø 3E BF ØE Ø7 75-1E 83 3E ØA 1C 64 7D 17

122Ø  E8 Ø8 Ø4 E8 98 F1 E8 78-Ø6 E8 27 Ø2 E8 AB Ø2 C6
123Ø  Ø6 BF ØE ØØ EB 2C 9Ø 8Ø-3E BF ØE Ø8 75 24 E8 EA
124Ø  Ø3 E8 CC F1 E8 ØØ Ø2 E8-FD Ø1 E8 FA Ø1 BA 42 19
125Ø  B4 Ø9 CD 21 E8 FC Ø1 E8-8Ø Ø2 C6 Ø6 BF ØE ØØ EB
126Ø  Ø1 9Ø 58 C3 8C C8 8E D8-8E CØ E8 B2 Ø3 BA CD 19
127Ø  B4 Ø9 CD 21 E8 2A Ø6 E8-F8 Ø7 A1 3F 12 A3 43 12
128Ø  A1 41 12 A3 45 12 B4 Ø2-CD 18 A8 Ø2 75 1C 8A 26
129Ø  Ø9 1C 22 E4 74 2D C6 Ø6-Ø9 1C ØØ 9Ø 5Ø E8 8B Ø3
12AØ  E8 BØ Ø1 E8 34 Ø2 58 EB-1A 9Ø 8A 26 Ø9 1C 22 E4
12BØ  75 11 C6 Ø6 Ø9 1C Ø1 9Ø-5Ø E8 6F Ø3 E8 94 Ø1 E8
```

```
12CØ   18 Ø2 58 A8 1Ø 74 24 B2-FF B4 Ø6 CD 21 5Ø AØ 3E
12DØ   12 22 CØ 75 1Ø BA B6 1A-B4 Ø9 CD 21 C6 Ø6 3E 12
12EØ   Ø1 9Ø E8 AC FD 58 E8 77-FD EB 8F A8 Ø1 75 Ø5 E8
12FØ   79 Ø4 EB 86 B2 FF B4 Ø6-CD 21 3C 34 75 33 A1 3F
13ØØ   12 48 79 ØF A1 41 12 48-79 Ø3 B8 ØØ ØØ A3 41 12
131Ø   B8 ØF ØØ A3 3F 12 A1 1Ø-1C A3 12 1C 48 78 Ø3 E9
132Ø   C4 ØØ A1 43 12 A3 3F 12-A1 45 12 A3 41 12 E9 49
133Ø   FF 3C 36 75 4A A1 3F 12-4Ø 3D 1Ø ØØ 7C 1E A1 41
134Ø   12 4Ø 3D Ø6 ØØ 7C ØF A1-43 12 A3 3F 12 A1 45 12
135Ø   A3 41 12 E9 24 FF A3 41-12 B8 ØØ ØØ A3 3F 12 A1

136Ø   1Ø 1C A3 12 1C 4Ø 3B Ø6-ØC 1C 7D Ø3 EB 78 9Ø 9Ø
137Ø   A1 43 12 A3 3F 12 A1 45-12 A3 41 12 E9 FB FE 3C
138Ø   32 75 39 A1 41 12 4Ø 3D-Ø6 ØØ 7C ØF A1 43 12 A3
139Ø   3F 12 A1 45 12 A3 41 12-E9 DF FE A3 41 12 A1 1Ø
13AØ   1C A3 12 1C Ø5 1Ø ØØ 3B-Ø6 ØC 1C 7C 39 A1 43 12
13BØ   A3 3F 12 A1 45 12 A3 41-12 E9 BE FE 3C 38 75 39
13CØ   A1 41 12 48 79 Ø3 B8 ØØ-ØØ A3 41 12 A1 1Ø 1C A3
13DØ   12 1C 2D 1Ø ØØ 79 ØF A1-43 12 A3 3F 12 A1 45 12
13EØ   A3 41 12 E9 94 FE A3 1Ø-1C E8 26 Ø1 E8 99 Ø1 BA
13FØ   17 1F B4 Ø9 CD 21 E9 81-FE 3C 2Ø 75 ØC E8 D1 Ø2

14ØØ   E8 6F Ø6 E8 B1 ØØ E9 71-FE 3C 1B 74 Ø3 E9 6A FE
141Ø   BA B8 2Ø B4 Ø9 CD 21 BA-D3 1D B4 Ø9 CD 21 B4 41
142Ø   CD 18 B8 ØØ ØC CD 21 2E-C5 16 38 12 B8 Ø6 25 CD
143Ø   21 BØ ØØ B4 4C CD 21 CF-ØØ ØØ ØØ ØØ ØA ØØ ØØ ØØ
144Ø   ØØ ØØ ØØ ØØ ØØ ØØ ØØ B9-ØØ ØØ 5Ø F7 EØ F7 EØ 58
145Ø   E2 F8 C3 B4 Ø2 CD 18 A8-Ø2 75 ØA A1 ØE 1C 2E A3
146Ø   ØC 1C EB Ø8 9Ø A1 ØA 1C-2E A3 ØC 1C B8 1Ø ØØ 83
147Ø   3E ØC 1C 1Ø 7D Ø7 8B ØE-ØC 1C EB Ø4 9Ø B9 1Ø ØØ
148Ø   A3 16 1C 33 CØ 51 5Ø E8-ØB Ø1 58 4Ø 83 Ø6 16 1C
149Ø   Ø4 59 E2 F1 8B ØE ØC 1C-3B C1 7D 1A 81 Ø6 16 1C

14AØ   CØ Ø9 81 3E 16 1C ØØ 3C-7F ØC 2B C8 83 F9 1Ø 7C
14BØ   D4 B9 1Ø ØØ EB CF C3 A1-38 1C 48 2B Ø6 32 1C Ø3
14CØ   Ø6 26 1C 8B 1E 28 1C F7-E3 8B 1E 2E 1C Ø3 C3 4Ø
14DØ   3B Ø6 2A 1C 7E Ø3 A3 2A-1C C3 C7 Ø6 1Ø 1C ØØ ØØ
14EØ   C7 Ø6 12 1C ØØ ØØ C7 Ø6-3F 12 ØØ ØØ C7 Ø6 41 12
14FØ   ØØ ØØ A1 41 12 BF ØØ ØA-F7 E7 8B 3E 3F 12 D1 E7
15ØØ   D1 E7 Ø3 F8 83 C7 1Ø 89-3E 14 1C E8 Ø4 ØØ E8 77
151Ø   ØØ C3 A1 14 1C A3 16 1C-A1 12 1C E8 77 ØØ Ø6 BE
152Ø   4D 19 B9 2Ø ØØ A1 41 12-BF ØØ ØA F7 E7 8B 3E 3F
153Ø   12 D1 E7 D1 E7 Ø3 F8 83-C7 1Ø 89 3E 14 1C 8B Ø4

154Ø   86 C4 BA ØØ B8 8E C2 26-Ø9 Ø5 F7 DØ BA ØØ A8 8E
155Ø   C2 26 21 Ø5 BA ØØ BØ 8E-C2 26 21 Ø5 8B 44 Ø2 86
156Ø   C4 BA ØØ B8 8E C2 26 Ø9-45 Ø2 F7 DØ BA ØØ A8 8E
157Ø   C2 26 21 45 Ø2 BA ØØ BØ-8E C2 26 21 45 Ø2 83 C6
158Ø   Ø4 83 C7 5Ø E2 B8 Ø7 C3-A1 3C 12 A3 16 1C E8 Ø1
159Ø   ØØ C3 A1 1Ø 1C 5Ø B4 Ø2-CD 18 A8 Ø2 58 74 13 57
15AØ   2E 8B 3E 16 1C 2E A2 6A-17 1E Ø6 E8 4C Ø4 Ø7 1F
15BØ   5F C3 1E Ø6 BB 8Ø Ø1 F7-E3 8B FØ B8 Ø4 ØA 8E D8
15CØ   2E 8B 3E 16 1C B9 2Ø ØØ-B8 ØØ A8 8E CØ 8B Ø4 26
15DØ   89 Ø5 8B 44 Ø2 26 89 45-Ø2 B8 ØØ BØ 8E CØ 8B 84

15EØ   8Ø ØØ 26 89 Ø5 8B 84 82-ØØ 26 89 45 Ø2 B8 ØØ B8
15FØ   8E CØ 8B 84 ØØ Ø1 26 89-Ø5 8B 84 Ø2 Ø1 26 89 45
16ØØ   Ø2 83 C7 5Ø 83 C6 Ø4 E2-BF Ø7 1F C3 57 2E 8B 3E
161Ø   16 1C 2E A2 6A 17 1E Ø6-E8 5Ø Ø3 Ø7 1F 5F C3 BB
162Ø   28 ØØ BA 9Ø Ø1 BD ØØ ØØ-EB ØA 9Ø BB 28 ØØ BA DØ
163Ø   ØØ BD ØØ ØØ 1E Ø6 B8 ØØ-ØØ 8B FD B9 ØØ A8 8E C1
164Ø   8B CB F3 AB 8B FD B9 ØØ-BØ 8E C1 8B CB F3 AB 8B
165Ø   FD B9 ØØ B8 8E C1 8B CB-F3 AB 83 C5 5Ø 4A 75 D9
166Ø   Ø7 1F C3 A1 1Ø 1C 1E Ø6-BB 8Ø Ø1 F7 E3 8B FØ A1
167Ø   32 1C BF ØØ ØA F7 E7 8B-3E 2E 1C D1 E7 D1 E7 Ø3

168Ø   F8 Ø3 3E 3A 1C B8 Ø4 ØA-8E D8 B9 2Ø ØØ B8 ØØ A8
169Ø   8E CØ 8B Ø4 26 89 Ø5 8B-44 Ø2 26 89 45 Ø2 B8 ØØ
16AØ   BØ 8E CØ 8B 84 8Ø ØØ 26-89 Ø5 8B 84 82 ØØ 26 89
16BØ   45 Ø2 B8 ØØ B8 8E CØ 8B-84 ØØ Ø1 26 89 Ø5 8B 84
16CØ   Ø2 Ø1 26 89 45 Ø2 83 C7-5Ø 83 C6 Ø4 E2 BF Ø7 1F
16DØ   C3 A1 38 1C 48 2B Ø6 32-1C Ø3 Ø6 26 1C 8B 3E 28
16EØ   1C F7 E7 8B 3E 2E 1C Ø3-F8 8A 26 Ø9 1C 22 E4 75
16FØ   11 1E B8 E4 1F 8E D8 2E-A1 1Ø 1C 88 Ø5 1F E8 62
17ØØ   FF C3 1E B8 E4 1F 8E D8-8A Ø5 A8 8Ø 75 2C 8Ø ØD
171Ø   8Ø 2E A1 1Ø 1C 88 85 ØØ-8Ø 1F A1 1Ø 1C A2 6A 17

172Ø   A1 32 1C BF ØØ ØA F7 E7-8B 3E 2E 1C D1 E7 D1 E7
173Ø   Ø3 F8 Ø3 3E 3A 1C E8 32-Ø2 C3 8Ø 25 7F 8A Ø5 2E
174Ø   A2 6A 15 C6 85 ØØ 8Ø ØØ-1F A1 32 1C BF ØØ ØA F7
175Ø   E7 8B 3E 2E 1C D1 E7 D1-E7 Ø3 F8 Ø3 3E 3A 1C 89
176Ø   3E 16 1C AØ 6A 15 E8 49-FE C3 ØØ B2 FF B4 Ø6 CD
177Ø   21 E8 A7 F9 3C 34 75 3C-A1 32 1C A3 34 1C A1 2E
178Ø   1C A3 3Ø 1C 48 79 27 A1-32 1C 48 79 1A A1 26 1C
179Ø   4Ø 3B Ø6 2C 1C 7C Ø4 A1-2C 1C 48 A3 26 1C E8 ØØ
```

```
17AØ   Ø1 B8 ØØ ØØ EB Ø8 9Ø A3-32 1C A1 36 1C 48 A3 2E
17BØ   1C E9 E9 ØØ 3C 36 75 49-A1 32 1C A3 34 1C A1 2E

17CØ   1C A3 3Ø 1C 4Ø 3B Ø6 36-1C 7C 3Ø A1 32 1C 4Ø 3B
17DØ   Ø6 38 1C 7C 2Ø A1 26 1C-48 79 Ø3 B8 ØØ ØØ A3 26
17EØ   1C E8 BD ØØ A1 38 1C 48-A3 32 1C A1 36 1C 48 A3
17FØ   2E 1C E9 A8 ØØ A3 32 1C-B8 ØØ ØØ A3 2E 1C E9 9C
18ØØ   ØØ 3C 32 75 2C A1 2E 1C-A3 3Ø 1C A1 32 1C A3 34
181Ø   1C 4Ø 3B Ø6 38 1C 7C 13-A1 26 1C 48 79 Ø3 B8 ØØ
182Ø   ØØ A3 26 1C E8 7A ØØ A1-38 1C 48 A3 32 1C EB 6D
183Ø   9Ø 3C 38 75 2C A1 2E 1C-A3 3Ø 1C A1 32 1C A3 34
184Ø   1C 48 79 17 A1 26 1C 4Ø-3B Ø6 2C 1C 7C Ø4 A1 2C
185Ø   1C 48 A3 26 1C E8 49 ØØ-B8 ØØ ØØ A3 32 1C EB 3D

186Ø   9Ø 3C 2Ø 75 ØC E8 69 FE-E8 Ø7 Ø2 E8 49 FC EB 3Ø
187Ø   9Ø 3C 1B 74 Ø1 C3 BA B8-2Ø B4 Ø9 CD 21 BA D3 1D
188Ø   B4 Ø9 CD 21 B4 41 CD 18-B8 ØØ ØC CD 21 2E C5 16
189Ø   38 12 B8 Ø6 25 CD 21 BØ-ØØ B4 4C CD 21 E8 D2 Ø1
18AØ   C3 2E C6 Ø6 69 17 ØØ 9Ø-8B 1E 26 1C A1 36 1C F7
18BØ   E3 8B D8 8B 16 38 1C BD-ØØ 73 1E Ø6 B8 E4 1F 8E
18CØ   D8 52 2E 8B ØE 36 1C 8B-FD 8A Ø7 3C FF 74 16 A8
18DØ   8Ø 74 12 8A A7 ØØ 8Ø 2E-88 26 6A 17 2E C6 Ø6 69
18EØ   17 Ø1 9Ø 24 7F B4 ØØ BE-8Ø Ø1 F7 E6 8B FØ 57 51
18FØ   1E B8 Ø4 ØA 8E D8 B9 2Ø-ØØ B8 ØØ A8 8E CØ 8B Ø4

19ØØ   26 89 Ø5 8B 44 Ø2 26 89-45 Ø2 B8 ØØ BØ 8E CØ 8B
191Ø   84 8Ø ØØ 26 89 Ø5 8B 84-82 ØØ 26 89 45 Ø2 B8 ØØ
192Ø   B8 8E CØ 8B 84 ØØ Ø1 26-89 Ø5 8B 84 Ø2 Ø1 26 89
193Ø   45 Ø2 83 C7 5Ø 83 C6 Ø4-E2 BF 1F 59 5F 2E 8Ø 3E
194Ø   69 17 ØØ 74 ØA E8 23 ØØ-2E C6 Ø6 69 17 ØØ 9Ø 43
195Ø   83 C7 Ø4 49 74 Ø3 E9 7Ø-FF 9Ø 81 ED ØØ ØA 5A 4A
196Ø   74 Ø3 E9 5C FF 9Ø Ø7 1F-C3 ØØ ØØ 53 51 52 57 1E
197Ø   Ø6 B8 64 13 8E D8 2E AØ-6A 17 B4 ØØ BB ØØ Ø2 F7
198Ø   E3 8B D8 B9 2Ø ØØ 8B 17-B8 ØØ A8 8E CØ 8B 87 8Ø
199Ø   ØØ 26 21 15 26 Ø9 Ø5 B8-ØØ BØ 8E CØ 8B 87 ØØ Ø1

19AØ   26 21 15 26 Ø9 Ø5 B8 ØØ-B8 8E CØ 8B 87 8Ø Ø1 26
19BØ   21 15 26 Ø9 Ø5 8B 57 Ø2-B8 ØØ A8 8E CØ 8B 87 82
19CØ   ØØ 26 21 55 Ø2 26 Ø9 45-Ø2 B8 ØØ BØ 8E CØ 8B 87
19DØ   Ø2 Ø1 26 21 55 Ø2 26 Ø9-45 Ø2 B8 ØØ B8 8E CØ 8B
19EØ   87 82 Ø1 26 21 55 Ø2 26-Ø9 45 Ø2 83 C3 Ø4 83 C7
19FØ   5Ø E2 93 Ø7 1F 5F 5A 59-5B C3 53 51 52 57 1E Ø6
1AØØ   B8 64 13 8E D8 2E AØ 6A-17 B4 ØØ BB ØØ Ø2 F7 E3
1A1Ø   8B D8 B9 2Ø ØØ B8 ØØ A8-8E CØ 8B 87 8Ø ØØ 26 89
1A2Ø   Ø5 B8 ØØ BØ 8E CØ 8B 87-ØØ Ø1 26 89 Ø5 B8 ØØ B8
1A3Ø   8E CØ 8B 87 8Ø Ø1 26 89-Ø5 8B 57 Ø2 B8 ØØ A8 8E

1A4Ø   CØ 8B 87 82 ØØ 26 89 45-Ø2 B8 ØØ BØ 8E CØ 8B 87
1A5Ø   Ø2 Ø1 26 89 45 Ø2 B8 ØØ-B8 8E CØ 8B 87 82 Ø1 26
1A6Ø   89 45 Ø2 83 C3 Ø4 83 C7-5Ø E2 AA Ø7 1F 5F 5A 59
1A7Ø   5B C3 2E C6 Ø6 69 17 ØØ-A1 34 1C BF ØØ ØA F7 E7
1A8Ø   Ø3 Ø6 3A 1C 8B 3E 3Ø 1C-D1 E7 D1 E7 Ø3 C7 A3 16
1A9Ø   1C A1 38 1C 48 2B Ø6 34-1C Ø3 Ø6 26 1C 8B 3E 28
1AAØ   1C F7 E7 8B 3E 3Ø 1C Ø3-F8 1E B8 E4 1F 8E D8 8A
1ABØ   Ø5 3C FF 74 15 A8 8Ø 74-11 8A A5 ØØ 8Ø 2E 88 26
1ACØ   6A 17 2E C6 Ø6 69 17 Ø1-24 7F 1F B4 ØØ E8 E2 FA
1ADØ   2E F6 Ø6 69 17 Ø1 74 Ø7-2E AØ 6A 17 E8 2D FB Ø6

1AEØ   BE 4D 19 B9 2Ø ØØ A1 32-1C BF ØØ ØA F7 E7 8B 3E
1AFØ   2E 1C D1 E7 D1 E7 Ø3 F8-Ø3 3E 3A 1C 8B Ø4 86 C4
1BØØ   BA ØØ B8 8E C2 26 Ø9 Ø5-BA ØØ A8 8E C2 26 Ø9 Ø5
1B1Ø   BA ØØ BØ 8E C2 26 Ø9 Ø5-8B 44 Ø2 86 C4 BA ØØ B8
1B2Ø   8E C2 26 Ø9 45 Ø2 BA ØØ-A8 8E C2 26 Ø9 45 Ø2 BA
1B3Ø   ØØ BØ 8E C2 26 Ø9 45 Ø2-83 C6 Ø4 83 C7 5Ø E2 BC
1B4Ø   Ø7 C3 1B 5B 33 3B 31 48-1B 5B 3Ø 4A 24 FF FF FF
1B5Ø   FF FF FF FF FF ØØ CØ Ø3-ØØ ØØ CØ Ø3 ØØ ØØ CØ Ø3
1B6Ø   ØØ ØØ CØ Ø3 ØØ ØØ CØ Ø3-ØØ ØØ CØ Ø3 ØØ ØØ CØ Ø3
1B7Ø   ØØ ØØ CØ Ø3 ØØ ØØ CØ Ø3-ØØ ØØ CØ Ø3 ØØ ØØ CØ Ø3

1B8Ø   ØØ ØØ CØ Ø3 ØØ ØØ CØ Ø3-ØØ ØØ CØ Ø3 ØØ ØØ CØ Ø3
1B9Ø   ØØ ØØ CØ Ø3 ØØ ØØ CØ Ø3-ØØ ØØ CØ Ø3 ØØ ØØ CØ Ø3
1BAØ   ØØ ØØ CØ Ø3 ØØ ØØ CØ Ø3-ØØ ØØ CØ Ø3 ØØ ØØ CØ Ø3
1BBØ   ØØ ØØ CØ Ø3 ØØ ØØ CØ Ø3-ØØ ØØ CØ Ø3 ØØ ØØ CØ Ø3
1BCØ   ØØ ØØ CØ Ø3 ØØ FF FF FF-FF FF FF FF FF Ø7 1B 5B
1BDØ   32 4A 1B 5B 3E 31 68 1B-5B 3E 35 68 CF AF CC DF
1BEØ   95 D2 8F 57 ØA ØD 56 31-2E 33 24 1B 5B 34 3B 32
1BFØ   48 95 D2 8F 57 81 4Ø 81-4Ø 81 4Ø 81 4Ø 81 4Ø 24
1CØØ   1B 5B 35 3B 32 48 4D 41-5Ø 2Ø CA DF CØ BØ DD A5
1C1Ø   DB BØ C4 DE 24 1B 5B 36-3B 32 48 4D 41 5Ø 2Ø CA

1C2Ø   DF CØ BØ DD A5 92 C7 89-C1 24 1B 5B 37 3B 32 48
1C3Ø   4D 41 5Ø 2Ø CA DF CØ BØ-DD A5 BE BØ CC DE 24 1B
1C4Ø   5B 38 3B 32 48 CF AF CC-DF A5 C3 DE BØ CØ A5 DB
1C5Ø   BØ C4 DE 24 1B 5B 39 3B-32 48 CF AF CC DF A5 C3
1C6Ø   DE BØ CØ A5 BE BØ CC DE-24 1B 5B 31 3Ø 3B 32 48
```

```
1C7Ø  93 47 2Ø CA DF CØ BØ DD-A5 DB BØ C4 DE 2Ø 24 1B
1C8Ø  5B 31 31 3B 32 48 93 47-2Ø CA DF CØ BØ DD A5 92
1C9Ø  C7 89 C1 2Ø 24 1B 5B 31-32 3B 32 48 93 47 2Ø CA
1CAØ  DF CØ BØ DD A5 BE BØ CC-DE 2Ø 24 1B 5B 37 3B 32
1CBØ  3Ø 3B 33 32 6D 24 1B 5B-37 3B 32 3Ø 3B 33 32 6D

1CCØ  1B 5B 33 3B 31 48 1B 5B-3Ø 4A 1B 29 33 98 95 95
1CDØ  95 95 95 95 95 95 95 95-95 95 95 95 99 1B 5B 34
1CEØ  3B 31 48 96 1B 29 3Ø 95-D2 8F 57 2Ø 2Ø 2Ø 2Ø 2Ø
1CFØ  2Ø 2Ø 2Ø 2Ø 2Ø 1B 29 33-96 1B 5B 35 3B 31 48 96
1DØØ  1B 29 3Ø 4D 41 5Ø 2Ø CA-DF CØ BØ DD A5 DB BØ C4
1D1Ø  DE 1B 29 33 96 1B 5B 36-3B 31 48 96 1B 29 3Ø 4D
1D2Ø  41 5Ø 2Ø CA DF CØ BØ DD-A5 92 C7 89 C1 1B 29 33
1D3Ø  96 1B 5B 37 3B 31 48 96-1B 29 3Ø 4D 41 5Ø 2Ø CA
1D4Ø  DF CØ BØ DD A5 BE BØ CC-DE 1B 29 33 96 1B 5B 38
1D5Ø  3B 31 48 96 1B 29 3Ø CF-AF CC DF A5 C3 DE BØ CØ

1D6Ø  A5 DB BØ C4 DE 1B 29 33-96 1B 5B 39 3B 31 48 96
1D7Ø  1B 29 3Ø CF AF CC DF A5-C3 DE BØ CØ A5 BE BØ CC
1D8Ø  DE 1B 29 33 96 1B 5B 31-3Ø 3B 31 48 96 1B 29 3Ø
1D9Ø  93 47 2Ø CA DF CØ BØ DD-A5 DB BØ C4 DE 2Ø 1B 29
1DAØ  33 96 1B 5B 31 31 3B 31-48 96 1B 29 3Ø 93 47 2Ø
1DBØ  CA DF CØ BØ DD A5 92 C7-89 C1 2Ø 1B 29 33 96 1B
1DCØ  5B 31 32 3B 31 48 96 1B-29 3Ø 93 47 2Ø CA DF CØ
1DDØ  BØ DD A5 BE BØ CC DE 2Ø-1B 29 33 96 1B 5B 31 33
1DEØ  3B 31 48 1B 29 33 9A 95-95 95 95 95 95 95 95 95
1DFØ  95 95 95 95 95 9B 1B 29-3Ø 24 1B 5B 3Ø 6D 1B 5B

1EØØ  33 3B 31 48 1B 5B 3Ø 4A-24 ØØ ØC ØØ ØC ØØ ØC ØØ
1E1Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
1E2Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-14 ØØ ØØ ØØ DC Ø5 ØØ ØØ
1E3Ø  ØØ ØØ Ø5 ØØ Ø5 ØØ 14 ØØ-Ø6 ØØ ØØ 41 ØD 1C 1B ØØ
1E4Ø  ØØ 3C ØØ 3B ØØ 3A ØØ 3D-56 ØC 99 ØB 42 ØB E6 ØA
1E5Ø  1Ø ØE 4B ØD E4 1F 8Ø 43-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
1E6Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
1E7Ø  7D ØØ 7D ØØ ØØ ØØ ØØ ØA-ØØ Ø4 ØØ ØØ ØØ ØØ ØØ 42
1E8Ø  3A 2A 2E 44 41 54 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
1E9Ø  2Ø 2Ø ØØ ØØ ØØ ØØ 42 3A-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø

1EAØ  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-ØØ ØØ ØØ ØØ 42 3A 2A 2E
1EBØ  2A 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø ØØ ØØ ØØ
1ECØ  ØØ 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
1EDØ  2Ø 2Ø ØØ ØØ 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
1EEØ  2Ø 2Ø 2Ø 2Ø 2Ø ØØ ØØ 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
1EFØ  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-ØØ ØØ 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
1FØØ  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø ØØ ØØ 2Ø 2Ø 2Ø
1F1Ø  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ ØØ
1F2Ø  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
1F3Ø  2Ø ØØ ØØ ØØ 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø

1F4Ø  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
1F5Ø  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
1F6Ø  2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
1F7Ø  2Ø 2Ø 2Ø 2Ø 1B 5B 3E 31-68 1B 5B 32 4A ØD ØA ØD
1F8Ø  ØA 24 1B 5B 73 1B 5B 31-3B 31 48 1B 5B 32 3Ø 3B
1F9Ø  33 32 6D 1B 5B 3E 35 68-ØØ 3D 3D 2Ø 4D 53 5Ø 54
1FAØ  45 52 2Ø 3D 3D 2Ø 28 43-29 31 39 38 39 2Ø 62 79
1FBØ  2Ø 54 2E 48 69 64 61 6B-61 2Ø 26 2Ø 4D 2E 41 6F
1FCØ  79 61 6D 61 2Ø 76 65 72-2E 31 2E 3Ø 1B 5B 4B 1B
1FDØ  5B 75 24 1B 5B 3E 35 6C-1B 5B 3E 31 6C ØD ØA 24

1FEØ  1B 5B 3E 35 68 1B 5B 33-3B 31 48 1B 5B 3Ø 4A 24
1FFØ  1B 5B 33 3B 31 48 1B 5B-31 4D 1B 5B 32 34 3B 31
2ØØØ  48 24 1B 5B 33 3B 31 48-1B 5B 31 4D 1B 5B 32 34
2Ø1Ø  3B 31 48 24 16 Ø7 1B 5B-32 32 3B 33 36 6D 5B 42
2Ø2Ø  5D 2Ø 1B 5B 32 33 3B 33-37 6D 1B 5B 3E 35 6C 24
2Ø3Ø  1B 5B 32 31 3B 33 33 6D-44 49 52 1B 5B 32 33 3B
2Ø4Ø  33 37 6D 1B 5B 3E 35 68-24 1B 5B 33 3B 31 48 1B
2Ø5Ø  5B 3Ø 4A 1B 5B 3E 35 68-24 1B 5B 33 3B 31 48 1B
2Ø6Ø  5B 32 31 3B 33 33 6D 5B-42 5D 1B 5B 32 32 3B 33
2Ø7Ø  36 6D 2Ø 4C 4F 41 44 2Ø-46 49 4C 45 2Ø 4E 41 4D

2Ø8Ø  45 2Ø 3A 2Ø 1B 5B 4B 1B-5B 32 33 3B 33 37 6D Ø7
2Ø9Ø  24 1B 5B 3E 35 68 1B 5B-3Ø 33 3B 31 48 1B 5B 32
2ØAØ  31 3B 33 33 6D 2Ø 2Ø 2A-1B 5B 32 32 3B 33 36 6D
2ØBØ  2Ø 53 41 56 45 2Ø 46 49-4C 45 2Ø 4E 41 4D 45 2Ø
2ØCØ  3A 2Ø 1B 5B 4B 1B 5B 32-33 3B 33 37 6D Ø7 1B 5B
2ØDØ  3E 35 6C 24 1B 5B 3E 35-68 1B 5B 3Ø 33 3B 31 48
2ØEØ  1B 5B 32 31 3B 33 33 6D-2Ø 2Ø 2A 1B 5B 32 32 3B
2ØFØ  33 36 6D 2Ø 4B 49 4C 4C-2Ø 46 49 4C 45 2Ø 4E 41
21ØØ  4D 45 2Ø 3D 2Ø 1B 5B 4B-1B 5B 32 33 3B 33 37 6D
211Ø  Ø7 1B 5B 3E 35 6C 24 Ø7-24 ØØ ØØ ØØ ØØ 1B 5B 3Ø

212Ø  33 3B 32 32 48 24 1B 5B-4B 24 1B 5B 3Ø 33 3B 32
213Ø  32 48 24 24 4C 4F 41 44-2Ø 45 4E 44 ØA ØD 24 42
214Ø  55 46 46 45 52 2Ø 46 55-4C 4C ØA ØD 24 2Ø 2Ø 2Ø
215Ø  2Ø 93 AF 82 B6 CC A7 B2-D9 82 AA 97 4C 82 E8 82
216Ø  DC 82 B7 81 42 91 B1 8D-73 82 B5 82 DC 82 B7 82
217Ø  A9 2Ø 59 2D 4E 2Ø 3F 24-2Ø 2Ø 2Ø 2Ø 8E CØ 8D 73
218Ø  8F 49 97 B9 24 1B 5B 32-31 3B 33 33 6D 2Ø 2Ø 2Ø
219Ø  2Ø B7 AC DD BE D9 21 21-1B 5B 32 33 3B 33 37 6D
21AØ  1B 5B 3E 35 68 24 2Ø 2Ø-2Ø 2Ø BE BØ CC DE 2Ø B4
21BØ  D7 BØ 21 21 24 2Ø 2Ø 2Ø-2Ø C3 DE A8 BD B8 B6 DE

21CØ  A4 B2 AF CA DF B2 C3 DE-BD 2Ø 21 21 24 1B 5B 3E
21DØ  35 68 41 3A 24 1B 5B 32-31 3B 33 33 6D 2Ø 2Ø 2Ø
21EØ  2Ø CC A7 B2 D9 2Ø B6 B8-C6 DD 2Ø 4F 4B 2Ø 59 2D
21FØ  4E 2Ø 3F 1B 5B 32 33 3B-33 37 6D 1B 5B 3E 35 68
22ØØ  24 2Ø 2Ø 2Ø 2Ø 4B 49 4C-4C 2Ø 2Ø B4 D7 BØ 21 21
221Ø  2Ø BC DE AF BA B3 A6 A4-B7 AC DD BE D9 2Ø BC CF
222Ø  BC CØ 24 2Ø 2Ø 2Ø 2Ø CC-A7 B2 D9 B6 DE B1 D8 CF
223Ø  BE DD 2Ø 21 21 24 Ø7 1B-5B 33 3B 31 48 1B 5B 3Ø
224Ø  4A 1B 5B 32 33 3B 33 37-6D CC A7 B2 D9 B6 DE B1
225Ø  D8 CF BE DD 2Ø ØD ØA 24-Ø7 1B 5B 3E 35 68 1B 5B

226Ø  32 33 3B 33 37 6D C3 DE-B2 BD B8 B6 DE BE AF C4
227Ø  ØD ØA BB DA C3 B2 CF BE-DD 24 1B 5B 34 3B 31 48
228Ø  1B 5B 31 4D 1B 5B 31 33-3B 31 48 24 1B 5B 31 33
229Ø  3B 31 48 1B 5B 3Ø 4B 1B-5B 34 3B 31 48 1B 5B 31
22AØ  4C 24 1B 5B 3E 35 6C 24-1B 5B 3E 35 68 24 1B 5B
22BØ  3Ø 34 3B 3Ø 31 48 24 24-1B 5B 31 3B 31 48 1B 5D
22CØ  32 31 3B 33 33 6D 54 4F-2Ø 4D 53 2Ø 44 4F 53 1B
22DØ  5B 32 33 3B 33 37 6D Ø7-24 1B 5B 32 33 3B 34 3Ø
22EØ  6D 24 1B 5B 32 33 3B 33-37 6D 24 ØØ ØØ ØØ ØØ ØØ
22FØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ

23ØØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
231Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
232Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
233Ø  C8 ØØ 96 18 F6
```

あとがき

マシン語によるゲームの制作はいかがでしたか，NECのPC-9801シリーズは，ゲーム専用機ではないため，ゲームに便利な特殊機能は用意されていませんが，すでに市販の商品が実証しているように，ゲームの分野でもかなりの力を発揮することができます．もちろん，漠然と本体を所持しているだけでは，既製のゲームソフトで遊ぶことしかできませんが…….

アレ，既製のゲームソフトで遊べれば，コンピュータでゲームをしたことになるんじゃないの？なんて思う方は，本書の読者の中にはいませんよね．この「あとがき」まで読むということは，すでにゲームの裏側に潜む《もうひとつのゲーム》に気が付いているはずですから．それとも，あるいは無意識のうちにここまで楽しんでしまったかもしれません．

そうです．ゲームには，遊ぶ楽しさと作る楽しさがあるのです．本書は，まさにこの作る側の楽しさを伝えるための本だったのです．現在，最も多く普及しているプログラム言語といえば，BASICやCといったいわゆる高級言語ですが，これらはハードウェアの特性や高速性を100%引き出すことができません．その代償として，プログラムの安全性や保守性を確保しているのですが，コンピュータは原発と違って最大の惨事でもマシンの暴走程度です．それも，リセットで簡単に現状復帰できるのですから，かわいいものじゃありませんか．マシンが火を噴いて爆発したり，プログラマを巻き込んだ環境汚染でもあるなら，暴走も「恐怖のかたまり」ですが，この程度なら「いたずら天使」といった存在です．

いたずら天使……？暴走を引き起こす主役に対して，余りにも可愛すぎる名称ですが，これは人間でいうなら痛みのようなものです．もしも痛みがなかったなら，ケガをしようと骨折しようと，それに気付いて手当をすることはできません．まさに暴走こそがプログラムを破滅から救っているといえるでしょう．そして，このバグの告知があるからこそ，われわれは思う存分マシン語で冒険をすることができるわけです．

こんな楽しい冒険の世界なのですが，とかくマシン語というと堅苦しいイメージがあります．たとえゲームが最終目的ではなくても，楽しいゲームを素材にすることで，マシン語に親しみを感じられるようになれたなら，きっとゲームの価値も見直されることでしょう．この先，どのようにマシン語を活用するか，それを決めるのは各自のアイデアとセンスです．ゲームに限らず，大いにマシン語を楽しんでほしいと思います．

ところで，本書には拡張されたプレーンである輝度面に関しての記述があまりありません．これは，対象となるPC-9801シリーズの機種を多くするための処置なのですが，プレーン数が増えればEGC（Enhanced Graphic Charger）についても言及したくなるからです．EGCを使った多重スクロールや上下左右のドット単位のスクロール……など，ページ数を無視すればテクニックのネタは尽きません．これらについては，いずれ近いうちに公開する予定となっていますので，グラフィック・テクニックに興味のある方は期待していてください．

まだまだ無限の可能性を秘めたマシン語の世界…….決して現状に満足することなく，夢とロマンを求めてチャレンジしてみてください．恐れるもの，失うものなど何もないのです．サァ，また新しい冒険の旅へ出かけようではありませんか．きっと，新たなゴールがあなたをやさしく迎えてくれるでしょう．

青山　学／日高　徹

最後に，本書を読む際の参考書として，あるいは本書を読み終えた方の次なるステップとして，以下の書籍をお勧めします．

『新版 PC-9800 シリーズテクニカルデータブック』　アスキー出版局テクライト編　著
『PC-9801 マシン語入門』　岩瀬正幸・藤井敬雄　共著
『はじめて読む MASM』　蒲地輝尚　著
『MS-DOS 3.1　ハンドブック』　アスキー出版局　編著
『実用 MS-DOS』　村瀬康治　著

以上 4 冊ともアスキー出版局刊

『ザ 8086 ブック』　吉川敏則　訳　産報出版刊
『8086 プログラミングデザイン』　山内　直　著　秀和システムトレーディング刊
『8086 マシン語秘伝の書』　日高　徹・青山　学　著　啓学出版刊
『PC-9800 シリーズはじめてのマシン語』　日高　徹・青山　学　共著　啓学出版刊

執筆者紹介

青山　学（あおやま　まなぶ）

1958年，東京生まれ．青山学院大学理工学部卒．

コンピュータ・エンジニアを経て，現在はフリーのゲームプログラマ．大型コンピュータからパソコンまで，言語・機種にこだわらないのを信条としている．日高氏との共著に「PC-9800シリーズ はじめてのマシン語」，「8086マシン語秘伝の書」(いずれも啓学出版）などがある．

日高　徹（ひだか　とおる）

1949年，栃木県宇都宮市生まれ．早稲田大学商学部卒．

PC-8801シリーズを中心に，オリジナルゲームを多数発表．代表作に「北斗の拳」(エニックス)，「ガンダーラ」(エニックス）などがある．

また，書籍の執筆も精力的に手掛けており，著書に「PC-8801mk II SRマシン語ゲーム・プログラミング」(アスキー出版局)，「マシン語ゲームグラフィックス」(小学館)，「Z80マシン語秘伝の書」(啓学出版)，青山氏との共著に「PC-9800シリーズ はじめてのマシン語」，「8086マシン語秘伝の書」(いずれも啓学出版）などがある．

PC-9801シリーズ
マシン語ゲーム・プログラミング

1991年2月21日　初版発行
1993年3月31日　第1版第6刷発行
定価2,500円（本体2,427円）

著　者　青山 学，日高 徹
発行者　藤井 章生
発行所　**株式会社アスキー**
〒107-24　東京都港区南青山6-11-1スリーエフ南青山ビル
振　替　東京4-161144
TEL (03)3486-7111（大代表）
第一書籍編集部 (03)3797-3225（ダイヤルイン）
出版営業部 TEL (03)3486-1977（ダイヤルイン）

制　作　株式会社 GARO
印　刷　凸版印刷株式会社

編　集　竹内充彦

ISBN4-7561-0062-7　　Printed in Japan

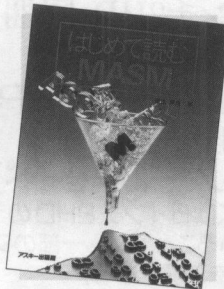
●11582

アスキーブックス
はじめて読む 8086

村瀬康治監修　蒲地輝尚著
A5判　定価1,650円（〒300）
（本体1,602円）

〈内容〉
マシン語から広がる世界／実行型ファイルをダンプする／実行型ファイルのメッセージを変更する／これだけは覚えて欲しいコンピュータの知識／8086CPUの基礎／マシン語命令の実習／やさしいプログラミングの実例／マクロアセンブラによるマシン語プログラミング／APPENDIX

アスキーブックス
はじめて読む MASM

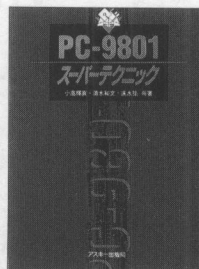
蒲地輝尚著
A5判　定価1,850円（〒300）
（本体1,796円）

〈内容〉
アセンブラをはじめる前に／アセンブラをとりまく世界／アセンブラ・プログラミングの基礎／セグメントの本格的活用／マクロアセンブラとモジュール別プログラミング／アセンブラ実用テクニック／APPENDIX

アスキー・ラーニングシステム
応用MS-DOS〈改訂新版〉

村瀬康治著
B5変　定価2,300円（〒300）
（本体2,233円）

〈内容〉
ユーザープログラム作成実習／MS-DOSのファイルシステム／MS-DOSの仕組みと働き／システムコールとソフトウェア割り込み／アセンブラによるソフトウェア開発／C言語によるプログラム開発デバイスドライバの作成とマウスの応用／APPENDIX

ソフトウェアバンク
PC-9801 スーパーテクニック

小高輝真・清水和文・速水祐　共著
B5判　定価4,200円(本体4,078円)

〈内容〉
CPU編／テキストVRAM編／グラフィック基礎編／GDC編／GRCG編／EGC編／キーボード編／シリアルポート編／割り込み編／その他周辺機器編

アスキー・ディスクアルバム

PC-9801シリーズ
マシン語ゲーム・プログラミング

青山 学, 日高 徹 共著

PC-9801版 3.5インチ2HD/5インチ2HD 同梱　定価 3,800円(〒400)

対象機種：PC-9801シリーズ(初代PC-9801とPC-9801Uは除く)
PC-98DO, DO+(98モード)
PC-98XL, XL², RL(ノーマルモード)
*ただし, 3.5インチ2HD, もしくは5インチ2HDのディスクが読めるフロッピーディスクドライブを最低1台は装備していること. また, 一部のプログラムでは, FM音源内蔵機種, またはFM音源ボード搭載機種を対象とする.

メモリ構成：640Kバイト以上

対象OS：MS-DOS Ver.2.11以降

〈内容〉
本書に掲載されたサンプルプログラムについて, ソースプログラムと実行形式プログラムを同時に収録. キャラクタやマップなどのデータファイルも収録しているため, お手持ちのマシンですぐに実行できます. Appendixのツールも実行形式プログラムを収録しました.

ウォーミング・アップ/キャラクタ・パターンの表示と移動/衝突と得点計算/音楽演奏と効果音/迷路型ゲーム/スクロール・ゲーム/Appendix

CHAPTER 1
ウォーミング・アップ

CHAPTER 2
キャラクタ・パターン
の表示と移動

CHAPTER 3
衝突と得点計算

CHAPTER 4
音楽演奏と効果音

CHAPTER 5
迷路型ゲーム

CHAPTER 6
スクロール・ゲーム

APPENDIX
ツールの入力
パターン・エディタ MSPTER
マップ・エディタ MAPEDIT

定価2,500円
(本体2,427円)

ISBN4-7561-0062-7 C3055 P2500E